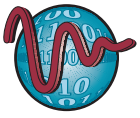


NATIONAL INSTRUMENTS™

Measurement Studio™

Measurement Studio LabWindows/CVI User Manual

NATIONAL INSTRUMENTS™

September 2001 Edition
Part Number 320681F-01

Worldwide Technical Support and Product Information

ni.com

National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 683 0100

Worldwide Offices

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 284 5011,
Canada (Calgary) 403 274 9391, Canada (Montreal) 514 288 5722, Canada (Ottawa) 613 233 5949,
Canada (Québec) 514 694 8521, Canada (Toronto) 905 785 0085, China (Shanghai) 021 6555 7838,
China (ShenZhen) 0755 3904939, Denmark 45 76 26 00, Finland 09 725 725 11, France 01 48 14 24 24,
Germany 089 741 31 30, Greece 30 1 42 96 427, Hong Kong 2645 3186, India 91805275406,
Israel 03 6120092, Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456, Malaysia 603 9596711,
Mexico 5 280 7625, Netherlands 0348 433466, New Zealand 09 914 0488, Norway 32 27 73 00,
Poland 0 22 528 94 06, Portugal 351 1 726 9011, Singapore 2265886, Spain 91 640 0085,
Sweden 08 587 895 00, Switzerland 056 200 51 51, Taiwan 02 2528 7227, United Kingdom 01635 523545

For further support information, see the *Technical Support Resources* appendix. To comment on the documentation, send e-mail to techpubs@ni.com.

Copyright © 1994, 2001 National Instruments Corporation. All rights reserved.

Important Information

Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this document is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

Trademarks

CodeBuilder™, CVI™, DataSocket™, IVI™, Measurement Studio™, National Instruments™, NI™, NI-488.2™, NI-DAQ™, NI-VXI™, ni.com™, and TestStand™ are trademarks of National Instruments Corporation.

Product and company names mentioned herein are trademarks or trade names of their respective companies.

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

Contents

About This Manual

Conventions xxiii
LabWindows/CVI Documentation Set xxiv
Standard Documentation Set xxiv
Print Documentation Sets xxiv
Related Documentation xxiv

Chapter 1
Configuring LabWindows/CVI

LabWindows/CVI Startup Options 1-1
How to Set the Configuration Options 1-2
Option Descriptions 1-3
Directory Options 1-3
cvidir 1-3
tmpdir 1-3
Date and Time Option—DSTRules 1-4
Timer Options—useDefaultTimer 1-4
Font Options 1-4
DialogFontName 1-4
DialogFontSize 1-4
DialogFontBold 1-4
MenuFontName 1-5
MenuFontSize 1-5
MenuFontBold 1-5

Chapter 2
LabWindows/CVI Overview

Components of LabWindows/CVI 2-1
Standard Libraries 2-2
Instrument Library 2-2
LabWindows/CVI Environment 2-3
How to Create Applications with LabWindows/CVI 2-4
Creating a User Interface 2-5
Creating Stand-Alone Programs and DLLs 2-5

Chapter 3
Project Window

Overview 3-1
Selecting Multiple Files in the Project Window 3-2
File Menu 3-3
File»New 3-3
File»Open 3-4
File»Save 3-4
File»Save Project As 3-4
File»Set Active Project 3-4
File»Save All 3-5
File»Auto Save Project 3-5
File»Print 3-5
File»Most Recently Closed Files 3-5
File»Exit LabWindows/CVI 3-5
Edit Menu 3-6
Edit»Workspace 3-6
Edit»Add Files to Project 3-6
Edit»Select All 3-7
Edit»Exclude File from Build/Include File in Build 3-7
Edit»Replace File in Project 3-7
Edit»Remove File 3-7
Edit»Move Item Up 3-7
Edit»Move Item Down 3-7
View Menu 3-8
View»Show Full Pathnames 3-8
View»Show Full Dates 3-8
View»Sort by Date 3-8
View»Sort by Name 3-8
View»Sort by Pathname 3-8
View»Sort by File Extension 3-8
View»No Sorting 3-8
Build Menu 3-8
Build»Configuration 3-8
Build»Create Debuggable Executable 3-9
Build»Create Debuggable Dynamic Link Library 3-9
Build»Create Release Executable 3-10
Build»Create Release Dynamic Link Library 3-10
Build»Create Static Library 3-10
Build»Batch Build 3-11
Build»Target Type 3-11

Build»Target Settings 3-12
Target Settings for Executables 3-12
Target Settings for DLLs 3-15
Target Settings for Static Libraries 3-20
Build»Compile File 3-20
Build»Mark File for Compilation 3-20
Build»Mark All for Compilation 3-21
Build»External Compiler Support 3-21
Build»Create Distribution Kit 3-23
Run Menu 3-31
Run»Debug 3-31
Run-Time Error Reporting 3-31
Debugging DLLs 3-31
Location of Files Required for Debugging DLLs 3-32
Different Ways to Debug DLLs 3-33
Running a Program in LabWindows/CVI 3-33
Running an External Process 3-33
Run»Continue 3-34
Run»Terminate Execution 3-34
Run»Break at First Statement 3-34
Run»Breakpoints 3-34
Run»Select External Process 3-35
Run»Execute 3-35
Run»Threads 3-35
Using Instrument Drivers 3-36
Instrument Driver Files 3-36
VXI*plug&play* Instrument Driver Files 3-37
Loading/Unloading Instrument Drivers 3-37
Precedence Rules for Loading the Instrument Driver Program File 3-37
Loading an Instrument without an Instrument Program 3-38
Modules that Contain Non-Instrument Functions 3-38
Modifying an Instrument Driver 3-39
Instrument Driver Fundamentals 3-39
Defining the Instrument Functions 3-40
Structuring Functions in an Instrument Driver 3-40
Defining the Hierarchy of Functions 3-41
Defining the Function Parameters 3-41
Data Types 3-41
Predefined Data Types 3-42
Intrinsic C Data Types 3-42
Meta Data Types 3-43

User-Defined Data Types ... 3-44
Creating a User-Defined Data Type ... 3-45
User-Defined Array Data Types ... 3-45
VISA Data Types ... 3-46
Input and Output Parameters ... 3-47
Return Values ... 3-47
Required Instrument Driver Functions ... 3-47
Building the Function Tree ... 3-48
Building the Function Panels ... 3-48
Writing the Function Code ... 3-49
Operating the Driver ... 3-49
Testing the Instrument Driver ... 3-49
Documenting the Driver ... 3-50
Instrument Menu ... 3-50
Instrument»Load ... 3-50
File Format Conversion ... 3-50
Instrument»Unload ... 3-50
Instrument»Edit ... 3-51
Accessing Function Panels from the Instrument Menu ... 3-51
Library Menu ... 3-52
Tools Menu ... 3-52
Tools»Create ActiveX Controller ... 3-52
Tools»Create ActiveX Server ... 3-53
Tools»Edit ActiveX Server ... 3-53
Tools»Create IVI Instrument Driver ... 3-53
Tools»Source Code Control ... 3-53
Tools»Source Code Browser ... 3-55
Browsing on Files ... 3-55
Browsing on Functions ... 3-56
Browsing on Variables, Data Types, and Macros ... 3-56
Tools»UI to Code Converter ... 3-56
Using the UI to Code Converter Utility ... 3-56
Using the Generated Code ... 3-57
UI to Code Converter Limitation ... 3-58
Tools»User Interface Localizer ... 3-58
Tools»Convert UI to Lab Style ... 3-59
User Defined Entries in the Tools Menu ... 3-59
Window Menu ... 3-59
Window»Cascade Windows ... 3-59
Window»Tile Windows ... 3-59
Window»Minimize All ... 3-60
Window»Close All ... 3-60
Window»Project ... 3-60
Window»Build Errors ... 3-60

Window»Run-Time Errors 3-60
Window»Debug Output 3-60
Window»Source Code Control Errors 3-61
Window»Memory Display 3-61
Window»Variables 3-61
Window»Watch 3-61
Window»Interactive Execution 3-61
Window»Open Source Files 3-62
Options Menu 3-62
Options»Build Options 3-62
Options»Compiler Defines 3-66
Predefined Macros 3-67
Options»Include Paths 3-68
Options»Instrument Directories 3-68
Options»Run Options 3-68
Options»Command Line 3-69
Options»Environment 3-69
Options»Library Options 3-71
National Instruments Libraries 3-71
User Libraries 3-72
Dummy .fp Files for Support Libraries 3-72
Options»Tools Menu Options 3-72
Options»Source Code Control Options 3-73
Options»Font 3-74
Options»Colors 3-75
Help Menu 3-75
Help»Contents 3-75
Help»Windows SDK 3-75
Help»Tip of the Day 3-76
Help»Web Links 3-76
Help»About LabWindows/CVI 3-76

Chapter 4
User Interface Editor Window

User Interface Editor Overview 4-1
Using the Pop-Up Menus of the User Interface Editor 4-2
CodeBuilder Overview 4-2
File Menu 4-3
File»New, Open, Save, and Exit LabWindows/CVI 4-3
File»Save As 4-3
File»Save Copy As 4-3
File»Close 4-3
File»Save All 4-3

File»Add File to Project ... 4-4
File»Read Only ... 4-4
File»Print ... 4-4
Edit Menu ... 4-4
Edit»Undo and Redo ... 4-4
Edit»Cut and Copy ... 4-5
Edit»Paste ... 4-5
Edit»Delete ... 4-5
Edit»Copy Panel and Cut Panel ... 4-5
Edit»Menu Bars ... 4-6
Edit»Panel ... 4-7
Edit»Control ... 4-8
Edit»Tab Order ... 4-9
Edit»Set Default Font ... 4-10
Edit»Apply Default Font ... 4-10
Edit»Control Style ... 4-10
Edit»Edit Custom Controls ... 4-10
Create Menu ... 4-10
Create»Panel ... 4-10
Create»Menu Bar ... 4-10
Create»Controls ... 4-10
View Menu ... 4-11
View»Find UIR Objects ... 4-11
View»Show/Hide Panels ... 4-12
View»Bring Panel to Front ... 4-12
View»Next Panel ... 4-12
View»Previous Panel ... 4-12
View»Preview User Interface Header File ... 4-12
Arrange Menu ... 4-13
Arrange»Alignment ... 4-13
Arrange»Align ... 4-13
Arrange»Distribution ... 4-14
Arrange»Distribute ... 4-14
Arrange»Control ZPlane Order ... 4-15
Arrange»Center Label ... 4-15
Arrange»Control Coordinates ... 4-15
Code Menu ... 4-15
Code»Set Target File ... 4-15
Code»Generate ... 4-15
Generate»All Code ... 4-16
Generate Main Function ... 4-17
Generate All Callbacks ... 4-18
Generate Panel Callbacks ... 4-18

Generate Control Callbacks 4-18
Generate Menu Callbacks 4-19
Code»View 4-19
Code»Preferences 4-20
Run Menu 4-20
Library Menu 4-21
Tools Menu 4-21
Window Menu 4-21
Options Menu 4-21
Options»Operate Visible Panels 4-21
Options»Next Tool 4-21
Options»Preferences 4-22
Options»Assign Missing Constants 4-23
Options»Save In Text Format 4-23
Options»Load From Text Format 4-23
Help Menu 4-23

Chapter 5
Source and Interactive Execution Windows

Source Windows 5-1
Toolbars in LabWindows/CVI 5-1
Modifying Your Toolbars 5-1
Adding and Positioning Buttons 5-2
Adding and Positioning Separators 5-2
Other Positioning Controls 5-2
Notification of External Modification 5-2
Context Menus 5-3
Interactive Execution Window 5-3
Using Subwindows 5-4
Selecting Text in the Source and Interactive Execution Window 5-4
File Menu 5-6
File»New, Open, Save, and Exit LabWindows/CVI 5-6
File»Open Quoted Text 5-6
File»Save As 5-6
File»Save Copy As 5-7
File»Close 5-7
File»Save All 5-7
File»Add File to Project 5-7
File»Read Only 5-7
File»Print 5-7
Edit Menu 5-7
Edit»Undo and Redo 5-8
Edit»Cut and Copy 5-8

Edit»Paste ... 5-8
Edit»Delete ... 5-9
Edit»Select All ... 5-9
Edit»Clear Window ... 5-9
Edit»Toggle Exclusion ... 5-9
Edit»Resolve All Excluded Lines ... 5-9
Edit»Insert Construct ... 5-9
Edit»Balance ... 5-10
Edit»Diff ... 5-10
Edit»Go To Definition ... 5-10
Edit»Go To Next Reference ... 5-11
Edit»Go Back ... 5-11
Edit»Find ... 5-11
Edit»Replace ... 5-14
Edit»Next File ... 5-14
View Menu ... 5-14
View»Line Numbers ... 5-14
View»Line Icons ... 5-15
View»Toolbar ... 5-15
View»Line ... 5-15
View»Beginning/End of Selection ... 5-15
View»Toggle Tag ... 5-15
View»Next Tag ... 5-15
View»Previous Tag ... 5-15
View»Tag Scope ... 5-16
View»Clear Tags ... 5-16
View»Function Panel Tree ... 5-16
View»Recall Function Panel ... 5-16
Invoking the Recall Function Panel Command ... 5-16
Recalling a Function Panel from a Function Name Only ... 5-17
Multiple Panels for One Function ... 5-17
Multiple Functions in One Function Panel Window ... 5-17
Syntax Requirements for the Recall Function Panel Command ... 5-17
View»Find Function Panel ... 5-17
View»Find UI Object ... 5-18
Build Menu ... 5-18
Build»Compile File ... 5-18
Build»Create Debuggable Executable ... 5-19
Build»Create Debuggable DLL ... 5-19
Build»Create Release Executable ... 5-19
Build»Create Release DLL ... 5-20
Build»Create Static Library ... 5-20
Build»Mark File for Compilation ... 5-20
Build»Clear Interactive Declarations ... 5-21

Build»Insert Include Statements....5-21
Build»Add Missing Includes....5-21
Build»Generate Prototypes....5-21
Build»Next/Previous Build Error....5-21
Build»Build Errors in Next File....5-21
Run Menu....5-22
Introduction to Breakpoints and Watch Expressions....5-22
Breakpoint State....5-22
Setting and Clearing Breakpoints....5-23
Conditional Breakpoints....5-23
Watch Expressions....5-23
Debug/Run Interactive Statements....5-24
Running in a Source Window....5-24
Running in the Interactive Execution Window....5-24
Run-Time Error Reporting....5-25
Run»Continue....5-25
Run»Go To Cursor....5-25
Run»Step Over....5-25
Run»Step Into....5-25
Run»Finish Function....5-25
Run»Terminate Execution....5-26
Run»Break at First Statement....5-26
Run»Toggle Breakpoint....5-26
Run»Breakpoints....5-26
Run»Stack Trace....5-27
Run»Up Call Stack....5-27
Run»Down Call Stack....5-27
Run»View Variable Value....5-27
Run»Add Watch Expression....5-27
Run»Threads....5-28
Instrument Menu....5-28
Library Menu....5-28
Tools Menu....5-29
Tools»Create ActiveX Controller....5-29
Tools»Create ActiveX Server....5-29
Tools»Edit ActiveX Server....5-29
Tools»Create IVI Instrument Driver....5-29
Tools»Edit Instrument Attributes....5-30
Tools»Edit Function Tree....5-30
Tools»Edit Function Panel....5-30
Tools»Source Code Control....5-30
Tools»Source Code Browser....5-30
Window Menu....5-31
Options Menu....5-31

Options»Editor Preferences 5-31
Options»Toolbar 5-31
Options»Bracket Styles 5-32
Options»Font 5-32
Options»Colors 5-32
Options»Syntax Coloring 5-32
Options»User Defined Tokens for Coloring 5-33
Options»Translate DOS LW Program 5-33
Options»Generate DLL Import Source 5-33
Options»Generate DLL Import Library 5-34
Options»Generate Visual Basic Include 5-34
Options»Create Object File 5-34
Options»Preprocess Source File 5-35
Help Menu 5-35

Chapter 6
Using Function Panels

Accessing Function Panels 6-2
Multiple Function Panels in a Window 6-3
Generated Code Box 6-3
Toolbars in LabWindows/CVI 6-4
Function Panel Controls 6-4
Specifying a Return Value Control Parameter 6-4
Specifying an Input Control Parameter 6-5
Specifying a Numeric Control Parameter 6-5
Specifying a Slide Control Parameter 6-5
Specifying a Binary Control Parameter 6-6
Specifying an Output Control Parameter 6-6
Using a Global Control 6-6
Common Control Function Panel 6-6
Convenient Viewing of Function Panel Variables 6-7
File Menu 6-7
File»New, Open, Save, and Exit LabWindows/CVI 6-7
File»Close 6-7
File»Save All 6-7
File»Add .FP File to Project 6-7
File»Add Program File to Project 6-7
File»Most Recently Closed Files 6-7
Code Menu 6-8
Code»Run Function Panel 6-8
Code»Declare Variable 6-8
Code»Clear Interactive Declarations 6-9
Code»Select UIR Constant 6-9

Code»Select Attribute Constant 6-10
Select Attribute Constant 6-10
Select Attribute Value 6-11
Code»Select Variable or Expression 6-12
What is Included in a List Box 6-12
Data Type Compatibility 6-13
Sorting List Box Entries 6-14
Code»Insert Function Call 6-14
Code»Set Target File 6-14
Code»View Variable Value 6-14
Code»Add Watch Expression 6-14
View Menu 6-15
View»Toolbar 6-15
View»Error 6-15
View»Include File 6-15
View»Current Tree 6-15
View»Function Panel History 6-15
View»Find Function Panel 6-15
View»Previous Function Panel 6-16
View»Next Function Panel 6-16
View»Previous Function Panel Window 6-16
View»Next Function Panel Window 6-16
View»First Function Panel Window 6-16
View»Last Function Panel Window 6-16
Instrument Menu 6-17
Library Menu 6-17
Tools Menu 6-17
Window Menu 6-17
Options Menu 6-18
Options»Default Control 6-18
Options»Default All 6-18
Options»Toolbar 6-18
Options»Exclude Function 6-18
Options»Toggle Control Style 6-18
Options»Change Format 6-19
Options»Edit Function Panel Window 6-19
Help Menu 6-19
Control 6-19
Function 6-19

Chapter 7
Function Tree Editor

About the Function Tree and Function Tree Editor 7-1
Function Tree Editor Context Menu 7-1
File Menu 7-2
File»New, Open, Save, and Exit LabWindows/CVI 7-2
File»Save .FP File 7-2
File»Save .FP File As 7-2
File»Save Copy of .FP File As 7-2
File»Close 7-2
File»Save All 7-3
File»Add .FP File To Project 7-3
File»Add Program File To Project 7-3
File»Read Only 7-3
Edit Menu 7-3
Edit»Cut 7-3
Edit»Copy 7-3
Edit»Paste Above 7-3
Edit»Paste Below 7-3
Edit»Edit Node 7-3
Edit»Edit Help 7-3
Edit»Edit Function Panel Window 7-4
Edit».FP Auto-Load List 7-4
Edit»Find 7-5
Edit»Replace 7-5
Create Menu 7-5
Create»Instrument 7-5
Create»Class 7-6
Adding a Class to an Empty Tree or Class 7-6
Create»Function Panel Window 7-6
Adding a Function to an Empty Tree or Class 7-7
Inserting a Function into an Existing Tree 7-7
Instrument Menu 7-7
Instrument»Load 7-7
Instrument»Unload 7-8
Instrument»Edit 7-8
Tools Menu 7-9
Tools»Create ActiveX Controller 7-9
Tools»Create ActiveX Server 7-9
Tools»Edit ActiveX Server 7-9
Tools»Create IVI Instrument Driver 7-9
Tools»Edit Instrument Attributes 7-9
Tools»Generate IVI C++ Wrapper 7-9

Tools»Enable Auto Replace 7-10
Tools»Generate New Source For Function Tree 7-10
Tools»Go To Definition 7-10
Tools»Go To Declaration 7-10
Tools»Source Code Control 7-10
Window Menu 7-10
Options Menu 7-11
Options».FP File Format 7-11
FP File Format Features 7-11
Options»Help Style 7-11
Options»Transfer Window Help to Function Help 7-12
Options»Generate Function Prototypes 7-12
Options»Generate Documentation 7-12
Options»Generate Windows Help 7-12
Options»Generate ODL File 7-12
Options»Generate DEF File 7-12
Options»Create DLL Project 7-13
Options»VXIplug&play Style 7-13
Help Menu 7-14
Function Tree Editor Examples 7-14
Creating a Function Tree with Multiple Classes 7-14
Cutting and Pasting Functions and Panels 7-15
Using Existing Function Panels in a New Driver 7-15
Editing Items in the Function Tree 7-16

Chapter 8
Function Panel Editor

Invoking the Function Panel Editor 8-1
Invoking from the Function Tree Editor 8-1
Invoking from a Function Panel 8-1
Function Panel Editor Menu Bar 8-1
File Menu 8-2
File»New, Open, Save, and Exit LabWindows/CVI 8-2
File»Save .FP File 8-2
File»Save .FP File As 8-2
File»Save Copy of .FP File 8-2
File»Close 8-3
File»Save All 8-3
File»Add .FP File to Project 8-3
File»Add Program File to Project 8-3
File»Read Only 8-3
Edit Menu 8-3
Edit»Undo and Redo 8-3

Edit»Cut Controls 8-4
Edit»Copy Controls 8-4
Edit»Paste 8-4
Edit»Cut Panel 8-4
Edit»Copy Panel 8-4
Edit»Edit Control 8-4
Edit»Change Control Type 8-5
Edit»Edit Function 8-5
Edit»Alignment 8-5
Edit»Align 8-5
Edit»Distribution 8-5
Edit»Distribute 8-5
Edit»Control Coordinates 8-5
Edit»Find 8-6
Edit»Replace 8-6
Edit»Control Help 8-6
Edit»Function Help or Window Help 8-7
Create Menu 8-7
Create»Input 8-7
Create»Slide 8-8
Create»Binary 8-8
Create»Ring 8-9
Create»Numeric 8-10
Create»Output 8-11
Create»Return Value 8-11
Create»Global Variable 8-12
Create»Message 8-12
Create»Function Panel 8-12
Create»Common Control Panel 8-13
View Menu 8-13
View»Toolbar 8-13
View»Error 8-13
View»Include File 8-13
View»Current Tree 8-13
View»Function Panel History 8-14
View»Find Function Panel 8-14
View»Previous Function Panel 8-14
View»Next Function Panel 8-14
View»Previous Function Panel Window 8-14
View»Next Function Panel Window 8-14
View»First Function Panel Window 8-15
View»Last Function Panel Window 8-15

Instrument Menu 8-15
Instrument»Load 8-15
Instrument»Unload 8-15
Instrument»Edit 8-16
Tools Menu 8-16
Tools»Create ActiveX Controller 8-16
Tools»Create ActiveX Server 8-16
Tools»Edit ActiveX Server 8-17
Tools»Create IVI Instrument Driver 8-17
Tools»Edit Instrument Attributes 8-17
Tools»Enable Auto Replace 8-17
Tools»Generate Source For Function Panel 8-17
Tools»Go To Definition 8-17
Tools»Go To Declaration 8-18
Window Menu 8-18
Options Menu 8-18
Options»Data Types 8-18
Options»Panels Movable 8-19
Options»Toolbar 8-19
Options»Initial Control Width 8-19
Options»Revert to Default Panel Size 8-19
Options»Toggle Scroll Bars 8-19
Options»Edit Function Tree 8-19
Options»Operate Function Panel 8-19
Help Menu 8-19
Function Panel Editor Examples 8-20
Creating a Function Window 8-20
Changing Control Type 8-23
Cutting and Pasting Controls 8-24

Chapter 9
Adding Help Information

New Style Versus Old Style Help 9-1
Help Options 9-2
Types of Help Information 9-2
Editing Help Information 9-2
Instrument Help 9-3
Function Class Help 9-3
Function Help (New Style Help Only) 9-3
Function Panel Window Help (Old Style Help Only) 9-4
Control Help 9-4

File Menu 9-5
File»New, Open, Save, and Exit LabWindows/CVI 9-5
File»Save .FP File 9-5
File»Save .FP File As 9-5
File»Save Copy of .FP File 9-5
File»Close 9-5
File»Save All 9-5
File»Add .FP File To Project 9-6
File»Add Program File To Project 9-6
File»Read Only 9-6
Edit Menu 9-6
Edit»Undo and Redo 9-6
Edit»Cut 9-6
Edit»Copy 9-6
Edit»Paste 9-6
Edit»Delete 9-6
Edit»Find 9-6
Edit»Replace 9-7
Edit»Revert 9-7
Tools Menu 9-7
Tools»Edit Function Panel 9-7
Tools»Edit Function Tree 9-7
Window Menu 9-7
Help Menu 9-7
Help Information Examples 9-8
Adding Help Information in the Function Tree Editor 9-8
Adding Help Information in the Function Panel Editor 9-9
Copying and Pasting Help Text 9-10

Chapter 10
Variables and Watch Windows

Variables Windows 10-1
Watch Window 10-2
File Menu 10-3
File»New, Open, Save, and Exit LabWindows/CVI 10-3
File»Output 10-3
File»Hide 10-3
File»Save All 10-3
File»Most Recently Closed Files 10-3
File»Exit LabWindows/CVI 10-3
Edit Menu for the Variables Window 10-3
Edit»Edit Value 10-3
Edit»Find 10-4

Edit»Next Scope 10-5
Edit»Previous Scope 10-5
Edit Menu for the Watch Window 10-5
Edit»Edit Value 10-5
Edit»Add/Edit Watch Expression 10-5
Edit»Delete Watch Expression 10-6
Edit»Find 10-6
View Menu 10-6
View»Expand Variable 10-7
View»Close Variable 10-7
View»Follow Pointer Chain 10-7
View»Retrace Pointer Chain 10-7
View»Go To Execution Position 10-8
View»Go To Definition 10-8
View»Source Code Browser 10-8
View»Array Display 10-8
View»String Display 10-8
View»Memory Display 10-8
View»Graphical Array View 10-9
1D Arrays 10-9
2D Arrays 10-11
Format Menu 10-12
Run Menu 10-12
Window Menu 10-13
Options Menu 10-13
Options»Variable Size 10-13
Options»Interpret As 10-13
Options»Estimate Number of Elements 10-14
Options»Add Watch Expression (Variables Window Only) 10-14
Help Menu 10-14

Chapter 11
Array and String Display Windows

Array Display Window 11-1
Multi-Dimensional Arrays 11-2
String Display Window 11-3
Multi-Dimensional String Array 11-3
File Menu 11-4
File»New, Open, Save, and Exit LabWindows/CVI 11-4
File»Output 11-4
File»Input 11-4
File»Close 11-4

File»Save All ... 11-4
File»Most Recently Closed Files ... 11-4
Edit Menu for the Array Display Window ... 11-5
Edit»Edit Value ... 11-5
Edit»Find ... 11-5
Edit»Goto ... 11-5
Edit Menu for the String Display Window ... 11-5
Edit»Edit Character ... 11-5
Edit»Edit Mode ... 11-5
Edit»Overwrite ... 11-6
Edit»Find ... 11-6
Edit»Goto ... 11-6
View Menu for the Array Display Window ... 11-6
View»Source Code Browser ... 11-6
View»String Display ... 11-6
View»Memory Display ... 11-6
View»Graphical Array View ... 11-7
Format Menu ... 11-7
Run Menu ... 11-7
Window Menu ... 11-7
Options Menu ... 11-8
Options»Reset Indices ... 11-8
Options»Display Entire Buffer ... 11-8
Help Menu ... 11-8

Appendix A
Source Window Keyboard Commands

Appendix B
Technical Support Resources

Glossary

Index

About This Manual

The *Measurement Studio LabWindows/CVI User Manual* contains detailed descriptions of LabWindows/CVI features and functionality. To use this manual effectively, you should be familiar with the *Getting Started with Measurement Studio LabWindows/CVI* manual, DOS, and Windows fundamentals.

Conventions

The following conventions appear in this manual:

» The **»** symbol leads you through nested menu items and dialog box options to a final action and through the Table of Contents in the LabWindows/CVI Help to a topic. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.

This icon denotes a note, which alerts you to important information.

This icon denotes a caution, which advises you of precautions to take to avoid injury, data loss, or a system crash.

bold Bold text denotes items that you must select or click on in the software, such as menu items and dialog box options. Bold text also denotes parameter names.

italic Italic text denotes variables, emphasis, a cross-reference, or an introduction to a key concept. This font also denotes text that is a placeholder for a word or value that you must supply. For references to the LabWindows/CVI Help, type the topic name in the index to access the topic.

`monospace` Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames and extensions, and code excerpts.

`monospace bold` Bold text in this font denotes the messages and responses that the computer automatically prints to the screen. This font also emphasizes lines of code that are different from the other examples.

`monospace italic`	Italic text in this font denotes text that is a placeholder for a word or value that you must supply.

LabWindows/CVI Documentation Set

Standard Documentation Set

The *Getting Started with Measurement Studio LabWindows/CVI* provides a hands-on introduction to LabWindows/CVI. This manual includes a tutorial that shows you how to develop applications in LabWindows/CVI.

The *LabWindows/CVI Help* contains comprehensive information on library functions, an in-depth look at the features available in LabWindows/CVI, information to help you develop programs in LabWindows/CVI, sample programs, and a tools library.

Print Documentation Sets

National Instruments offers two options for printed documentation for LabWindows/CVI. The first package includes the *Measurement Studio LabWindows/CVI User Manual*, the *Measurement Studio LabWindows/CVI Instrument Driver Developers Guide*, and the *Measurement Studio LabWindows/CVI Programmer Reference Manual*.

The second package includes the *Measurement Studio LabWindows/CVI Library Reference Volume 1*, the *Measurement Studio LabWindows/CVI Reference Volume 2*, and the *Measurement Studio User Interface and Attribute Reference*.

Related Documentation

- *Measurement Studio LabWindows/CVI Programmer Reference Manual*
- *Measurement Studio LabWindows/CVI Instrument Driver Developers Guide*
- *LabWindows/CVI Help*
- *NI-488.2 Function Reference Manual for Windows*
- *NI-DAQ User Manual for PC Compatibles*
- *NI-DAQ Function Reference Help*
- *NI-VXI online help*
- *NI-VISA User Manual*

- *NI-VISA Programmer Reference Manual*
- *OLE 2 Programmers Reference, Volume 2*

1

Configuring LabWindows/CVI

This chapter describes special options that override some of the configuration defaults established during the LabWindows/CVI installation or through the configuration dialog boxes within the environment.

These options inform LabWindows/CVI where to find system files, where to place temporary files, and so on. You might not need to set any of these options.

Getting Started with Measurement Studio LabWindows/CVI contains installation instructions for LabWindows/CVI and a hands-on tutorial. It is a good idea to be familiar with the material in *Getting Started with Measurement Studio LabWindows/CVI* before you go through *Measurement Studio LabWindows/CVI User Manual*.

LabWindows/CVI Startup Options

You can append certain options to the `cvi` command line, separating various parameters by spaces. The valid startup options appear in the following table.

Option	Purpose
`<filename>`	LabWindows/CVI automatically loads the file at startup. The file can be any of the types available under the **File»Open** command in LabWindows/CVI.
`-run`	This option automatically invokes the **Debug** command from the **Run** menu of LabWindows/CVI.
`-run_then_exit`	This option automatically invokes **Run»Debug** and then automatically invokes **File»Exit LabWindows/CVI** when the project terminates. This option also suppresses the LabWindows/CVI startup screen and Project window.
`-newproject`	LabWindows/CVI starts with an empty Project window.
`-p`*`ProcessID`*	LabWindows/CVI attaches to the process that *`ProcessID`* identifies. When the process subsequently loads DLLs, LabWindows/CVI can debug them if you created them with debugging information in LabWindows/CVI. You can express *`ProcessID`* as a decimal number or as a hexadecimal number that you precede with `0x`.

How to Set the Configuration Options

LabWindows/CVI Development Environment Configuration Options are under the following key, where `[version]` is the version of the LabWindows/CVI Development Environment:

```
HKEY_LOCAL_MACHINE\SOFTWARE\National Instruments\CVI\[version]
```

For example, use the following key to set the configuration options for the LabWindows/CVI 6.0 Development Environment:

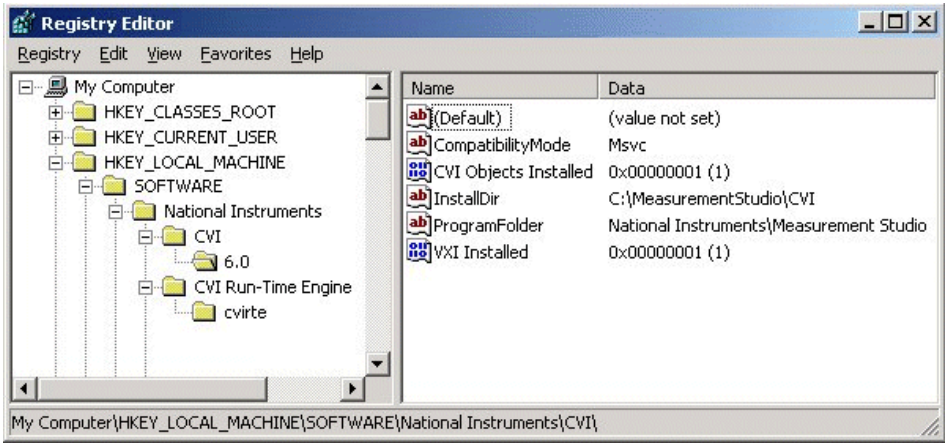
```
HKEY_LOCAL_MACHINE\SOFTWARE\National Instruments\CVI\5.5
```

LabWindows/CVI Run-time Engine configuration options are under the following key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\National Instruments\CVI Run-Time
Engine\cvirte
```

Your programs, when you run them from the environment or standalone, use the Run-Time Engine configuration options.

A configuration string value is associated with each option, as shown in the following figure.



You do not have to include an unused configuration string in the Registry.

You must specify an absolute pathname, including a drive letter, for configuration strings that take a directory name.

Option Descriptions

When configuring LabWindows/CVI, you can make changes to the directory options, date and time options, timer options, and the font options.

Directory Options

This section contains detailed descriptions of the directory options available in LabWindows/CVI.

cvidir

You have to set the `cvidir` option only if the subdirectories that LabWindows/CVI requires, shown in the following table, are not in the directory that contains the LabWindows/CVI executable. The `cvidir` option specifies the directory that contains the subdirectories.

Name of Directory	Contents
`bin`	Resource files (`cvi.rsc`, `cvimsgs.txt`), National Instruments function panels (`.lfp files`), National Instruments libraries (`.obj` and `.lib`).
`font`	Font description files.
`include`	C header files for National Instruments libraries.
`sdk`	Windows SDK.

If you do not specify a directory, LabWindows/CVI assumes that the directory that contains the executable file `cvi.exe` or `cvi` also contains the directories in the preceding table.

tmpdir

`tmpdir` sets the location for temporary files.

If you do not specify a directory, LabWindows/CVI uses the value of the environment variable `TMP`. If the value of `TMP` is not defined or is invalid, LabWindows/CVI uses the value of the environment variable `TEMP`. If the value of `TEMP` is not defined or is invalid, LabWindows/CVI uses the directory that contains `cvi.exe`.

If you run LabWindows/CVI across a network, you must set `tmpdir` to one of your local directories.

Date and Time Option—DSTRules

The DSTRules option allows you to specify the portions of the year daylight savings time is in effect in your area. This affects ANSI C Library functions such as mktime and localtime. Refer to the *Library Reference»ANSI C Library»Time and Date Functions* topic in the **LabWindows/CVI Help** for more information.

Timer Options—useDefaultTimer

If you set the useDefaultTimer option to True, LabWindows/CVI uses the default Windows timer to implement the LabWindows/CVI timing-related functions, such as Timer and Delay. The default Windows timer provides a resolution of 55 ms under Windows 98/95 and 10 ms under Windows 2000/NT.

If you set useDefaultTimer to False under Windows 98/95, LabWindows/CVI uses the Windows multimedia library timer. The multimedia library timer provides a resolution of 1 ms.

If you set useDefaultTimer to False under Windows 2000/NT, LabWindows/CVI attempts to use the performance counter timer. The performance counter timer provides a resolution of 1 ms. If the performance counter timer is not available, LabWindows/CVI uses the multimedia library timer, which provides a resolution of 1 ms.

The default value for useDefaultTimer is False.

Font Options

Under Windows, LabWindows/CVI provides configuration options to set the fonts that LabWindows/CVI uses in dialog boxes and menus.

DialogFontName

DialogFontName specifies the font that LabWindows/CVI uses in dialog boxes and the built-in pop-up panels, as in the following example, DialogFontName=Courier.

DialogFontSize

DialogFontSize specifies the font size that LabWindows/CVI uses in dialog boxes and the built-in pop-up panels, as in the following example, DialogFontSize=30.

DialogFontBold

DialogFontBold specifies whether the font that LabWindows/CVI uses in dialog boxes and the built-in pop-up panels is bold, as in the following example, DialogFontBold=Yes.

MenuFontName

`MenuFontName` specifies the font that LabWindows/CVI uses in menus, as in the following example, `MenuFontName=Courier`.

MenuFontSize

`MenuFontSize` specifies the font size that LabWindows/CVI uses in menus, as in the following example, `MenuFontSize=11`.

MenuFontBold

`MenuFontBold` specifies whether the font that LabWindows/CVI uses in menus is bold, as in the following example, `MenuFontBold=Yes`.

2

LabWindows/CVI Overview

Components of LabWindows/CVI

LabWindows/CVI is a programming environment for developing instrument control, automated test, and data acquisition applications in ANSI C.

LabWindows/CVI has the following components.

- Standard libraries and interactive function panels for the following components:
  - GPIB
  - RS-232
  - VISA (Virtual Instrument Software Architecture)
  - Data acquisition (distributed with National Instruments PC-based data acquisition boards)
  - Data analysis
  - Transport Control Protocol (TCP)
  - DataSocket
  - Windows Dynamic Data Exchange (DDE) communication
  - File I/O
  - Data formatting
  - ANSI C
- A graphical User Interface Editor, CodeBuilder Wizard, and library for building, displaying, and controlling a graphical user interface
- A wizard and library for controlling ActiveX servers
- A wizard and library for creating IVI instrument drivers, which are highly structured VXI*plug&play*-compatible instrument drivers that use an attribute model to enable advanced features, such as state-caching, simulation, and compatibility with generic instrument classes
- A set of instrument drivers that contains high-level functions and interactive function panels for controlling specific instruments
- A development environment with windows to manage projects and source code with complete editing, debugging, and user-protection features

An additional library, the Advanced Analysis Library, is available for LabWindows/CVI. This library is an optional package that you can order from National Instruments.

Standard Libraries

LabWindows/CVI includes the following standard libraries:

- User Interface Library
- Analysis Library
- Easy I/O for DAQ Library
- Data Acquisition Library
- GPIB-488/488.2 Library
- RS-232 Library
- VISA Library
- IVI Library
- TCP Library
- DataSocket Library
- DDE Library
- ActiveX Library
- Formatting and I/O Library
- Utility Library
- ANSI C Library

The functions that make up these libraries can be executed in the LabWindows/CVI environment. Refer to the following manuals:

- *NI-VISA User Manual* (available upon request)
- *NI-VISA Programmer Reference Manual* (available upon request)

Instrument Library

The Instrument Library is a set of instrument drivers that each contain high-level C functions for controlling a specific GPIB, RS-232, or VXI instrument. The high-level functions encapsulate the low-level steps necessary to control the instrument and read data. You can use instrument drivers in the environment in the same way you use the other LabWindows/CVI libraries.

LabWindows/CVI Environment

The LabWindows/CVI environment makes it easy for you to create and test applications that use the LabWindows/CVI libraries. The environment is a combination editor, compiler, and debugger with extensive run-time checking. A special feature called a *function panel* makes the task of developing programs much easier. Using a function panel, you can execute a LabWindows/CVI library function interactively and generate code that calls the function. Function panels also contain online help information for the functions and function parameters. You can build, execute, test, and debug the source code for your application in the LabWindows/CVI environment.

The LabWindows/CVI environment also has a User Interface Editor for creating a graphical user interface for your application programs. You can control the user interface using functions in the User Interface Library.

Also, you can use the LabWindows/CVI environment to create instrument drivers.

The LabWindows/CVI environment has the following windows, each with its own menu bar.

- Project window—This window appears when you start LabWindows/CVI. You use this window to open, edit, build, run, and save application project (`.prj`) files. A project file is a list of files your application uses. Chapter 3, *Project Window,* describes the Project window in detail.
- User Interface Editor windows—You use these windows to build graphics-mode command bars, pull-down menus, dialog boxes, controls, graphs, and strip charts and save them to User Interface Resource (`.uir`) files. Chapter 4, *User Interface Editor Window,* describes the User Interface Editor window in detail.
- Source windows—You use these windows to create, edit, run, debug, and save source code. These windows include an optional toolbar to give you quick access to commands you use frequently. Chapter 5, *Source and Interactive Execution Windows,* describes the Source window in detail.
- Interactive Execution window—You use this window to execute selected portions of code. You do not have to have a complete program in the Interactive Execution window, as is the case in a Source window. For instance, you can execute variable declarations and assignment statements in C without declaring a `main` function. Refer to Chapter 5, *Source and Interactive Execution Windows*, for more information.
- Function Panels—You use these windows to interactively execute library functions and insert code into the Source window. These windows include an optional toolbar to give you quick access to commands you use frequently. Chapter 6, *Using Function Panels,* describes the Function Panel window in detail.
- Function Tree Editor windows—You use these windows to build the tree structure of function panel files. Refer to Chapter 7, *Function Tree Editor*, for more information.

- Function Panel Editor windows—You use these windows to build function panels. These windows include an optional toolbar to give you quick access to commands you use frequently. Refer to Chapter 8, *Function Panel Editor,* for more information.
- Function Tree Help Editor and Function Panel Help Editor windows—You use these windows to add online help to function panels. Refer to the Chapter 9, *Adding Help Information,* for more information on these windows.
- Variables, Array Display, String Display, and Watch windows—You use these windows for debugging programs. Refer to Chapter 10, *Variables and Watch Windows*, and Chapter 11, *Array and String Display Windows*, for more information.

You develop applications in the LabWindows/CVI environment using the ANSI C programming language. LabWindows/CVI complies with the ANSI X3.159-1989 and the ISO/IEC 9899:1990 standards for the C programming language.

For information on the LabWindows/CVI compiler/linker and how to use the LabWindows/CVI libraries with other compilers and linkers, refer to Chapter 3, *Compiler/Linker Issues,* in the *Measurement Studio LabWindows/CVI Programmer Reference Manual*.

How to Create Applications with LabWindows/CVI

Use LabWindows/CVI as a text editor in which to enter your entire program. You can greatly simplify application development by using function panels to execute LabWindows/CVI functions and to automatically insert the code into your program. Function panels contain complete online help. Refer to Chapter 6, *Using Function Panels*, for more details.

The Project window contains all the component files of your application. The simplest case is one source file.

A typical project, however, contains multiple code modules and a User Interface Resource file. You can list code modules as source files or compiled files. You can debug source files, and LabWindows/CVI performs run-time error checking when you execute code in source files.

To include compiled files, such as library or object files, in your project, you must compile them with LabWindows/CVI or a compatible external compiler. Refer to Chapter 3, *Compiler/Linker Issues,* in the *Measurement Studio LabWindows/CVI Programmer Reference Manual* for more information on compatible external compilers. Compiled files consume less memory and run faster than source files. However, you cannot debug them, and they do not have run-time error checking.

You can mark a source file in the project list to be compiled without debugging to use less memory.

You can strike a balance between initial project start-up time, execution speed, memory consumption, and the ability to debug code modules by varying the types of code modules you list in your project.

Creating a User Interface

You can create user interface objects (panels, controls, menus, and so on) using the User Interface Editor window and save them in a `.uir` file. You can load, display, and modify these objects in your program using the functions in the User Interface Library. Also, you can specify callback functions that LabWindows/CVI calls when events occur on these objects.

The LabWindows/CVI CodeBuilder automatically generates complete C code that compiles and runs based on a user interface (`.uir`) file you create or edit. By choosing certain options presented to you in the **Code** menu, you can produce *skeleton code*. Skeleton code is syntactically and programmatically correct code that can compile and run before you type a single line of code. With the CodeBuilder feature, you save the time of typing standard code you must include in every program, eliminate syntax and typing errors, and maintain an organized source code file with a consistent programming style. For more information, refer to the *CodeBuilder* topic in the **LabWindows/CVI Help**.

Creating Stand-Alone Programs and DLLs

With the LabWindows/CVI Run-time Engine, you can create standalone executables, dynamic link libraries, and static libraries. Refer to Chapter 4, *Creating and Distributing Stand-Alone Executables and DLLs*, in the *Measurement Studio LabWindows/CVI Programmer Reference Manual,* for more information.

3

Project Window

Overview

Use the Project window to open, edit, build, run, and save application project (`.prj`) files. A project file is a list of files your application uses. Certain files must be in the list, while others are optional. If you had a project loaded the last time you used LabWindows/CVI, that project appears in the Project window when you start LabWindows/CVI again.

Unless you use the following files as instrument driver program files or load them dynamically using `LoadExternalModule`, you must put them in your project file list.

- Source files your application program uses, ending with `.c`
- Object files your application program uses, ending with `.obj`
- Library files your application program uses, ending with `.lib` (DLL import libraries are in this category.)

The following files are optional in your project file list.

- Header files (`.h`) your application program uses. Listing `.h` files makes it easy to open them for viewing or editing and ensures that the compiler can find them.
- User interface resource (`.uir`) files your application program uses. Listing `.uir` files makes it easy to open them for viewing or editing and ensures that LabWindows/CVI can find them.
- Instrument driver function panel (`.fp`) files. Listing `.fp` files lets LabWindows/CVI automatically load instruments when you open the project.
- Instrument driver program files. Listing these files overrides the loading precedence for instrument driver program files. Refer to *Using Instrument Drivers* and *Instrument Menu* sections for information about instrument driver program files.

You can open `.c`, `.h`, and `.uir` files in the project list by double-clicking directly on the filename. Double-clicking on a `.fp` filename opens the Select Function Panel dialog box for the instrument driver.

You can use the following icons in the Project window.

The file is currently closed. Double-click on this icon to open the file. If the file is a `.fp` file, double-clicking opens the Function Tree Editor.

The file is currently open. Double-click on this icon to close the file. If the file is a `.fp` file, double-clicking hides the Function Tree Editor window, but does not unload the `.fp` file.

S

The file has been modified since you last saved it. Double-click on this icon to save the file.

C

You have modified the file since you last compiled it, or you manually marked it for compilation. Double-click on this icon to compile the file.

O

This icon applies only to source (`.c`) files and indicates that you enabled the **Compile Without Debugging** option. If you enable this option, LabWindows/CVI compiles the source file without debugging information. You can use this option to reduce the amount of memory required to build a project. Double-click on this icon to toggle the option.

|

The file is associated with a loaded instrument driver.

A

This icon indicates that the `.fp` file is loaded into the **Instrument** menu and signifies that it is attached to or associated with a program file.

U

This icon indicates that the `.fp` file is loaded into the **Instrument** menu and signifies that it is unattached to any program file. When you double-click on this icon, LabWindows/CVI tries to attach a program file.

If no icon appears in the I column next to a `.fp` file, the `.fp` file is not loaded into memory. When you double-click on the U icon, LabWindows/CVI tries to load the `.fp` file into memory and attach the instrument driver program file.

☑

The box in this icon appears whenever the file is in the current source code control system project. A checkmark appears in the box if the file is currently checked out. Double-click on the box to check the file in or out of the source code control system.

Selecting Multiple Files in the Project Window

You can execute commands on multiple files in the project by selecting multiple files and then executing the command through its hot key or menu item. Use one or more of the following methods to select multiple files in the project window.

- <Ctrl-Left-Click>—Adds a file to the currently selected files.
- <Shift-Left-Click>—Adds the files between the last selected file and the file you click on to the currently selected files.
- <Ctrl-Shift-A>—Selects all the files in the project.

- <Shift-Down Arrow>—Adds the file below the currently selected file to the list of selected files.
- <Shift-Up Arrow>—Adds the file above the currently selected file to the list of selected files.

File Menu

This section explains how to use the commands in the **File** menu.

File»New

Use the **New** command to open various types of new empty windows.

If you choose **Source** or **Include**, a new Source window appears in which you can create a new `.c` or `.h` file.

If you choose **User Interface**, a new User Interface Editor window appears in which you can create a new `.uir` file. Refer to Chapter 4, *User Interface Editor Window*, for more information about User Interface Editor windows.

If you choose **Project**, a dialog box appears with a message that asks if you want to unload the current project. Note that you can work with only one project at a time. If you select **Yes**, LabWindows/CVI prompts you to save any modified files in the old project. After you save or discard changes, the New Project Options dialog box appears. To include the new project in the existing workspace, select **Create Project in Current Workspace**. To add the project to a new workspace, select **Create Project in New Workspace**. LabWindows/CVI stores in the workspace the settings that do not affect the way your project builds. If you do not add the new project to the existing workspace, LabWindows/CVI creates a new workspace for the new project. You also are prompted to transfer project options from the old project to the new project. The *Options Menu* section describes these options.

If you choose **Workspace**, a dialog box appears with a message that asks if you want to unload the current workspace and its associated project(s). Select **Yes** to unload the current workspace. LabWindows/CVI stores in the workspace the settings that do not affect the way your project builds. Because LabWindows/CVI requires you to name a workspace as soon as you create it, you must use the **Save** button in the next dialog box that appears to name and write the workspace to disk.

If you choose **Function Tree**, a new Function Tree Editor window appears in which you can create a new `.fp` file. Refer to the Chapter 7, *Function Tree Editor*, for more information about Function Tree Editor windows.

File»Open

Use the **Open** command to open various types of user specified files. When you select **Open**, a dialog box appears, prompting you for a filename to load into a new window. One feature of this dialog box is the Directories ring, which is at the top of the dialog box. When activated, this ring displays a list of directories from which you have opened files previously.

If you choose **Source** or **Include**, a Source window appears with your specified `.c` or `.h` file.

If you choose **User Interface**, a User Interface Editor window appears with your specified `.uir` file.

If you choose **Project**, a Project window appears with your specified `.prj` file. LabWindows/CVI prompts you to save any modified files in the old project. If you open a project that does not have a workspace of the same name, LabWindows/CVI creates a new workspace for the project with default options. National Instruments recommends that you open workspaces.

Choose the **Workspace** command to open an existing workspace from the list in the dialog box. LabWindows/CVI stores in the workspace the settings that do not affect the way your project builds. Opening the workspace that contains the project you want to open is an alternative to opening the project file directly.

If you choose **Function Tree**, a new Function Tree Editor window appears with your specified `.fp` file. You cannot use this command to load instrument modules. You load instrument modules from the **Instrument** menu.

File»Save

Use the **Save** command to write the project (`.prj`) file to disk. If you want to append a different extension, type it in after the filename. If you do not want to append any extension, enter a period after the filename.

File»Save Project As

Use the **Save As** command to write the project file to disk using a new name you specify. The **Save As** command changes the name on the Project window title bar to the new name you specified. If you want to append an extension other than `.prj`, type it in after the filename. If you do not want to append any extension, enter a period after the filename.

File»Set Active Project

Use the **Set Active Project** command to open a project from the current workspace.

File»Save All

The **Save All** command saves all open files to disk.

File»Auto Save Project

If you enable the **Auto Save Project** command, LabWindows/CVI automatically saves your project files. When you load a project, the **Auto Save Project** command is initially enabled unless the project file is read only on disk. If you enable the command, LabWindows/CVI automatically saves the project file whenever the project contains significant new or modified information. If you disable this command, the project file is saved only in the following cases:

- When you execute the **Save**, **Save As**, or **Save All** command from the **File** menu.
- When you unload the project or exit LabWindows/CVI. LabWindows/CVI prompts you to save the file in this case.

Notice that if you disable the **Auto Save Project** command, LabWindows/CVI does not save the project file when you start running a program, even if you set the **Save Changes before Running** option in the Run Options dialog box to Always or Ask.

File»Print

The **Print** command opens a list of all the printable files in the project. You can select the files you want to print.

File»Most Recently Closed Files

For your reference, two lists appear in the **File** menu.

- A list of the four most recently closed files, other than project files
- A list of the four most recently closed workspace or project files

File»Exit LabWindows/CVI

Use the **Exit LabWindows/CVI** command to close the current LabWindows/CVI session. If you have modified any open files since the last save, or if any windows contain unnamed files, LabWindows/CVI prompts you to save them.

Edit Menu

This section explains how to use the commands in the Project window **Edit** menu.

Edit»Workspace

You can include multiple projects in a workspace. LabWindows/CVI stores in the workspace the settings that do not affect the way your project builds. When you create a new project, a dialog box prompts you to add the new project in the existing workspace or to create a new workspace for the project.

The Edit Workspace dialog box displays the project files you included in your workspace. The Edit Workspace dialog box has the following options:

- **Add**—Adds a project to the workspace. You can browse to the project you want to add.
- **Remove**—Removes a project from the workspace.
- **Move Up**—Moves the selected project up one line. This list determines the order in which projects appear in **File»Set Active Project**.
- **Move Down**—Moves the selected project down one line.
- **OK**—Accepts your changes and closes the dialog box.
- **Cancel**—Closes the dialog box without accepting any changes.
- **Help**—Displays help for workspaces.

Edit»Add Files to Project

Use the **Add Files to Project** item to add any type of file to the project list. Choose any one of the file types listed in the menu to invoke a dialog box and then select a file from the dialog box.

Use the **Source**, **Object**, and **Library** items to add code modules to your project. Refer to Chapter 2, *Using Loadable Compiled Modules*, in the *Measurement Studio LabWindows/CVI Programmer Reference Manual* for information about using object, library, and DLL files in LabWindows/CVI.

An import library (`.lib`) file must accompany each DLL. If you want to use a DLL in your project, you must list the import library rather than the DLL. You cannot add DLL and DLL path (`.pth`) files to the project. If you load a project that was created in Windows 3.1 and contains `.dll` or `.pth` files, LabWindows/CVI displays a warning message and excludes the files.

For more detailed information on using DLLs in LabWindows/CVI, refer to Chapter 3, *Compiler/Linker Issues,* in the *Measurement Studio LabWindows/CVI Programmer Reference Manual*.

Use the **Include** item to add header files to your project. It is a good idea to list header files in your project because this makes access to your header files much easier.

Use the **User Interface** item to add `.uir` files to your project. You can list `.uir` files in your project to make access to the files easier.

Use the **Instrument** command to add instrument drivers to your project. Instrument drivers that LabWindows/CVI loads through the project remain in memory while the project is open.

Use the **All Files** command to add any file to your project. The **All Files** command brings up the Add Files to Project dialog box and lists all files available in the selected directory.

Edit»Select All

Use the **Select All** command to select all of the files in the project. You select specific files using the keyboard and mouse as described in the *Selecting Multiple Files in the Project Window* section.

Edit»Exclude File from Build/Include File in Build

The **Exclude File from Build** command excludes the highlighted code module file from the build. This command does not apply to `.h`, `.fp`, or `.uir` files. Excluded files appear in a different color in the Build window. LabWindows/CVI does not compile or link them into the project. When you exclude a file, the command toggles to **Include File in Build** so you can include the file in the build again.

Edit»Replace File in Project

Use the **Replace File in Project** command to replace the selected files in your project. In the dialog box that appears, browse to a file and add the new file to the project.

Edit»Remove File

Use the **Remove File** command to remove the selected files from the project list.

Edit»Move Item Up

Use the **Move Item Up** command to move the selected files up one line in the project list. To activate this menu item, select **No Sorting** from the **View** menu.

Edit»Move Item Down

Use the **Move Item Down** command to move the selected files down one line in the project list. To activate this menu item, select **No Sorting** from the **View** menu.

View Menu

This section explains how to use the commands in the **View** menu.

View»Show Full Pathnames

Use this command to toggle between displaying the project list with full pathnames and displaying the project list with simple base filenames.

View»Show Full Dates

Use this command to toggle between displaying the project list with short file dates, such as `08/20/01`, and full file dates, such as `Monday, August 20, 2001`.

View»Sort by Date

If you sort by date, the project list appears in chronological order.

View»Sort by Name

If you sort by name, the project list appears in alphabetical order by filename.

View»Sort by Pathname

If you sort by pathname, the project list appears in alphabetical order by directory pathname.

View»Sort by File Extension

If you sort by file extension, the project list appears in alphabetical order by file extension.

View»No Sorting

If you choose **No Sorting**, you can list your project files in any order by using the **Move Item Up** and **Move Item Down** items in the **Edit** menu.

Build Menu

This section explains how to use the commands in the Project window **Build** menu. Use commands in the **Build** menu for compiling files, building and linking projects, marking files for compilation, and creating application files.

Build»Configuration

The **Configuration** item opens a submenu in which you select the active configuration for your project. Set the configuration to **Debug** when you want to debug your executable or

DLL. Set the configuration to **Release** when you are ready to build a release (stand-alone) version of your executable, DLL, or static library. If you choose **LabVIEW RealTime Support** as run-time support in the Target Settings dialog box, the **Debug** configuration is dimmed.

Note You can set the names of the target executable, DLL, or library files for each configuration using the **Target Settings** menu item.

When you select the **Release** item in the **Configuration** submenu, source modules execute faster, but you sacrifice the ability to set breakpoints or use the Variables window. Also, you have no protection from run-time memory errors such as using bad pointers, over-indexing arrays, passing incorrect array sizes, and so on.

Build»Create Debuggable Executable

The **Create Debuggable Executable** menu item appears if you select the **Debug** item in the **Configuration** submenu and the **Executable** item in the **Target Type** submenu.

Use this menu item to compile and build an executable with debugging information. Use the **Debugging level** control on the Build Options dialog box to select the degree of debugging information you want LabWindows/CVI to generate for the executable. For information on the **Debugging level** control settings, refer to the *Options»Build Options* section. To debug the executable you create with this command, use the **Debug** menu item in the **Run** menu of a Project, Source, or Variables window.

You can specify the filename of the executable, as well as other executable settings, by selecting the **Target Settings** menu item. You can set other compile and run options using the menu items in the **Options** menu of the Project window.

Build»Create Debuggable Dynamic Link Library

The **Create Debuggable Dynamic Link Library** menu item appears if you select the **Debug** item in the **Configuration** submenu and the **Dynamic Link Library** item in the **Target Type** submenu.

Use this menu item to compile and build a DLL with debugging information. Use the **Debugging level** control on the Build Options dialog box to select the degree of debugging information you want LabWindows/CVI to generate for the executable.

You can specify the filename of the DLL, as well as other DLL settings, by selecting the **Target Settings** menu item. You can set other compile and run options using the menu items in the **Options** menu of the Project window.

The **Debug** command also generates a DLL import library for the DLL. For information on debugging DLLs, refer to the *Debugging DLLs* section.

Build»Create Release Executable

The **Create Release Executable** menu item appears if you select the **Release** item in the **Configuration** submenu and the **Executable** item in the **Target Type** submenu.

Use this menu item to compile and build an executable without debugging information. This command ignores the value of the **Debugging level** control on the Build Options dialog box.

You can specify the filename of the executable, as well as other executable settings, by selecting the **Target Settings** menu item. You can set other compile and run options using the menu items in the **Options** menu of the Project window.

Build»Create Release Dynamic Link Library

The **Create Release Dynamic Link Library** menu item appears if you select the **Release** item in the **Configuration** submenu and the **Dynamic Link Library** item in the **Target Type** submenu.

Use this menu item to compile and build a DLL without debugging information. This command ignores the value of the **Debugging level** control on the Build Options dialog box.

You can specify the filename of the DLL, as well as other DLL settings, by selecting the **Target Settings** menu item. You can set other compile and run options using the menu items in the **Options** menu of the Project window.

This command also generates a DLL import library for the DLL.

Build»Create Static Library

The **Create Static Library** menu item appears if you select the **Release** item in the **Configuration** submenu and the **Static Library** item in the **Target Type** submenu. The **Release** item is always checked.

Use this menu item to compile and build a static library without debugging information. This command ignores the value of the **Debugging level** control on the Build Options dialog box.

You can specify the filename of the static library, as well as other static library settings, by selecting the **Target Settings** menu item. You can set other compile and run options using the menu items in the **Options** menu of the Project window.

If you include a `.lib` file in a static library project, LabWindows/CVI includes all object modules from the `.lib` in the static library. This differs from creating an executable or DLL, in which LabWindows/CVI includes only the `.lib` modules that other modules in the project reference. In addition, LabWindows/CVI reports an error if you attempt to build a static library when you have a DLL import library in your project.

Build»Batch Build

Use the Batch Build dialog box to build multiple projects.

- **Configurations list box**—Displays all the valid configurations for the projects in your workspace.
- **Check All Debug**—Selects all the debug configuration items in the list.
- **Check All Release**—Selects all the release configuration items in the list.
- **Check None**—Deselects all the items in the list.
- **Build**—Switches to and builds each selected configuration of each project.
- **Rebuild**—Forces LabWindows/CVI to recompile all source files in each selected configuration of each project.
- **Cancel**—Cancels the operation and removes the Batch Build dialog box.

Build»Target Type

The **Target Type** item opens a submenu in which you select the target type for your project. The target type determines what type of file you create when you execute the **Create** command that appears below **Configuration** in the **Build** menu. The name of the **Create** command changes depending on the target type and configuration you select. You can select from the following target types:

- Executable
- Dynamic Link Library
- Static Library

The **Executable** and **Static Library** options are dimmed if you select **LabVIEW RealTime Only** as the run-time support in the Target Settings dialog box.

When you select **Static Library**, the **Debug** command in the **Run** menu is dimmed. If you select **Dynamic Link Library**, you can use the **Select External Process** command in the **Run** menu to specify an external program that uses the DLL. When you do this, the **Debug** command changes to **`Debug xxx.exe`**, where ***`xxx.exe`*** is the name of the program you specify.

Build»Target Settings

The **Target Settings** item brings up the Target Settings dialog box. The Target Settings dialog box contains different controls depending on the item you check in **Target Type** submenu.

Target Settings for Executables

When you set the **Target Type** to **Executable** and select **Build»Target Settings**, the Target Settings dialog box has the following options:

- **Application File**—The names of the executable files for the debug and release versions of your program. Change the value of the ring control to edit the filenames for the debug and release configurations. You can use the **Browse** button to select an existing filename.
- **Application Title**—A descriptive title for your program. This title appears in the **Start** menu of Windows if you create a distribution kit using the **Create Distribution Kit** command in the **Build** menu. The operating system registers it when a user starts the application.
- **Application Icon File**—A file that contains a descriptive graphical icon for your program. You can double-click on the icon in the Windows shell to start your executable. You can use the **Browse** button to select an existing icon file.
- **Icon**—The graphical representation of the **Application Icon File**. You can double-click on this control to browse for an icon file on disk. The sample program `samples\apps\iconedit.exe` ships with LabWindows/CVI so that you can create your own icon files.
- **Runtime Support**—You can choose the runtime support for your stand-alone executable in this ring control. If you select the **Instrument Driver Only** item, your project does not link to the entire set of LabWindows/CVI libraries but instead links to a smaller set of functions.

  Stand-alone executables you create with the **Full Runtime Engine** item selected use the LabWindows/CVI Run-time Engine DLL, `cvirte.dll`. Stand-alone executables you create with the **Instrument Driver Only** item selected use `instrsup.dll` instead of `cvirte.dll`.

  If you use a stand-alone compiler and want to use `instrsup.dll`, include `cvi\extlib\instrsup.lib` in your external compiler project instead of `cvirt.lib` and `cvisupp.lib`. Remember that when you use an external compiler, you link to that compiler's ANSI C library.

  `instrsup.dll` contains functions from the following libraries:

  – Formatting and I/O Library (Except `ArrayToFile` and `FileToArray`)

  – RS-232 Library

– Utility Library (selected functions only; refer to the Utility Library Functions discussion later in this section)
– ANSI C Library

Your project also can link to the following libraries:

– Analysis or Advanced Analysis Library
– GPIB Library
– VXI Library
– VISA Library
– IVI Library
– Data Acquisition Library—If your project files call the `Config_Alarm_Deadband`, `Config_ATrig_Event_Message`, `Config_DAQ_Event_Message`, `DIG_Change_Message_Config`, `Get_DAQ_Event`, or `Peek_DAQ_Event` functions in the Data Acquisition Library, you will get link errors when you build your program in LabWindows/CVI. The link errors occur because the preceding functions use the LabWindows/CVI User Interface Library internally, which is not available in the `instrsup` and `cvi_lvrt` run-time engine DLLs. Your project files can call any of the other functions in the Data Acquisition Library.

If you use a stand-alone compiler and want to use any of these libraries, refer to Chapter 3, *Compiler/Linker Issues,* in the *Measurement Studio LabWindows/CVI Programmer Reference Manual*.

If you use the **Create Distribution Kit** command on a project that you link for instrument driver support only, LabWindows/CVI automatically includes `instrsup.dll` in the distribution kit and disables the option to distribute the full LabWindows/CVI Run-time Engine.

The following Utility Library functions are in `instrsup.dll`.

– `Beep`
– `DateStr`
– `Delay`
– `SyncWait`
– `Timer`
– `TimeStr`
– `RoundRealToNearestInteger`
– `TruncateRealNumber`
– `Instand-aloneExecutable`
– `CVIRTEHasBeenDetached`

`instrsup.dll` does not support the Standard Input/Output window. Functions such as `FmtOut` or `ScanIn` return errors when you use them with `instrsup.dll`.

All the functions in `instrsup.dll` are multithread safe.

- **Create Console Application**—If you leave this box unchecked, your executable is created as a Windows GUI application. If you check this box, your executable is created as a console application. Console applications create a Windows console window (Command Prompt or MS-DOS Prompt) and set the standard I/O port to the console. Refer to `SetStdioPort` for information on standard I/O port options. Create a console application if you want to redirect the standard input or output of your program.
- **Version Info**—When you click on this button, the Version Info dialog box appears. You can enter version information for the executable file in this dialog box. LabWindows/CVI saves the version information in the executable as a standard Windows version resource. You can obtain the information from the executable by using the Windows SDK functions `GetFileVersionInfo` and `GetFileVersionInfoSize`.

  In the Version Info dialog box, the entries for **File Version** and **Product Version** must be in the form:

  ```
  n, n, n, n
  ```

  where *n* is a number from 0 to 255.
- **LoadExternalModule Options**—The following options assist you in loading external modules.
  - Enable LoadExternalModule—Select this option if your project uses `LoadExternalModule`. This option is enabled by default. If your project does not use `LoadExternalModule`, disable this option to reduce the size of your executable. If you disable the option but still use `LoadExternalModule`, LabWindows/CVI prompts you to enable full support. You must rebuild your project before the changes take effect.
  - Add Files to Executable—Use this button to select additional module files you want to link into the Application File. These are modules that your project files do not directly reference but that are referenced by modules you load at runtime by calling `LoadExternalModule`.
  - If you force a Windows SDK import library into your project, your executable might not start up successfully. The Windows SDK import libraries contain functions that are not present on all versions of Windows. If the DLL on your system does not export all the functions in the import library, your executable will fail at startup. Instead of forcing an import library into your executable, you can force only the functions you need into the executable. To force specific functions into your executable, create a table of function pointers, and add the functions to the table. For example, to force references to `CreateWindow` and `GetFreeDiskSpace`, you can add the following code to a source file in your project.

    ```
    void* ReferenceFunctionsTable[] = {
    CreateWindow, GetDiskFreeSpace,
    }
    ```

- **OK**—This button accepts the current inputs and closes the dialog box.
- **Cancel**—This button cancels the operation and removes the dialog box.

Target Settings for DLLs

When you set the **Target Type** to **Dynamic Link Library** and select **Build»TargetSettings**, the Target Settings dialog box has the following options:

- **DLL File**—The name of the DLL files for the debug and release versions of your program. Changing the value of the ring control allows you to edit the filename for the debug and release configurations. You can use the **Browse** button to select an existing filename.

Note If you select the **LabVIEW Real-Time Only** option for **Runtime Support**, the filename you select must conform to the 8.3 DOS standard to be downloadable to a LabVIEW Real-Time board.

- **Import Library Base Name**—Normally, the name of the import library is the same as the name of the DLL except that the extension is `.lib`. There might be some cases, however, where you want to use a different name. For example, you might want to append `_32` to the name of your DLL to distinguish it as a 32-bit DLL but not append it to the import library name. This is, in fact, the convention used for VXI*plug&play* instrument driver DLLs. If you want to enter a different name for the import library, deselect the **Use Default** option. Enter a name without any directory names.
- **Where to Copy DLL**—Use this ring control to instruct LabWindows/CVI to copy the DLL to a different directory after creating it. Your choices are the following:
  - Do not copy
  - Windows System directory
  - VXI*plug&play* directory (the `bin` directory under the VXI*plug&play* framework directory)
- **Runtime Support**—You can choose the run-time support for your DLL in this ring control. If you select the **Instrument Driver Only** item or the **LabVIEW Real-Time Only** item, your project does not link to the entire set of LabWindows/CVI libraries but instead links to a smaller set of functions.

  DLLs you create with the **Full Runtime Engine** item selected use the LabWindows/CVI Run-time Engine DLL, `cvirte.dll`. DLLs you create with the **Instrument Driver Only** item selected use `instrsup.dll` instead of `cvirte.dll`. The **Instrument Driver Only** option is particularly useful for creating instrument driver DLLs. It allows other applications to use instrument driver DLLs without having to load the large LabWindows/CVI Run-time Engine DLL. DLLs you create with the **LabVIEW Real-Time Only** item selected use `cvi_lvrt.dll` instead of `cvirte.dll` or

instrsup.dll. The **LabVIEW Real-Time Only** option is useful for creating DLLs that will be downloaded and run on the LabVIEW Real-Time hardware.

If you use a stand-alone compiler and want to use instrsup.dll, include cvi\extlib\instrsup.lib in your external compiler project instead of cvirt.lib and cvisupp.lib. Remember that when you use anexternal compiler, you link to that compiler's ANSI C library.

instrsup.dll contains functions from the following libraries:

– Formatting and I/O Library
– RS-232 Library (Except InstallComCallback)
– Utility Library (selected functions only; refer to the Utility Library Functions discussion later in this section)
– ANSI C

Your project also can link to the following libraries:

– Analysis or Advanced Analysis Library
– GPIB Library
– VXI Library
– VISA Library
– IVI Library
– Data Acquisition Library—If your project files call the Config_Alarm_Deadband, Config_ATrig_Event_Message, Config_DAQ_Event_Message, DIG_Change_Message_Config, Get_DAQ_Event, or Peek_DAQ_Event functions in the Data Acquisition Library, you will get link errors when you build your program in LabWindows/CVI. The link errors occur because the preceding functions use the LabWindows/CVI User Interface Library internally, which is not available in the instrsup and cvi_lvrt run-time engine DLLs. Your project files can call any of the other functions in the Data Acquisition Library.

If you use a stand-alone compiler and want to use any of these libraries, refer to Chapter 3, *Compiler/Linker Issues,* in the *Measurement Studio LabWindows/CVI Programmer Reference Manual*.

If you use the **Create Distribution Kit** command on a project that you link for instrument driver support only, LabWindows/CVI automatically includes instrsup.dll in the distribution kit and disables the option to distribute the full LabWindows/CVI Run-time Engine.

The following Utility Library functions are in instrsup.dll.

- Beep
- DateStr
- Delay
- SyncWait

- Timer
- TimeStr
- RoundRealToNearestInteger
- TruncateRealNumber
- InstandaloneExecutable
- CVIRTEHasBeenDetached

instrsup.dll does not support the Standard Input/Output window. Functions such as FmtOut or ScanIn return errors when you use them with instrsup.dll.

All the functions in instrsup.dll are multithread safe.

cvi_lvrt.dll is a subset of instrsup.dll and has the following additional restrictions:

- The RS-232 Library is not available.
- The file I/O and console I/O functions in the ANSI C, Formatting and I/O, and Utility libraries are not supported and will return run-time errors. Some of the functions affected by this are printf, scanf, fprintf, fscanf, ScanIn, ScanFile, FmtOut, FmtFile, Beep, tmpfile, and tmpnam.
- Only the "C" locale is recognized, and all case-sensitive functions such as stricmp and tolower only perform conversions from characters a–z to characters A–Z, and vice-versa, and only perform comparisons using this set of characters. Locale specific case-sensitivity is not honored.
- DLLs that link to cvi_lvrt.dll can be built only in the Release configuration, and you cannot debug them.

You can link your **LabVIEW Real-Time Only** project to the VISA library, the Data Acquisition library, or the NI-CAN library. If you do so, however, you must not combine calls to these LabWindows/CVI libraries with calls to the corresponding libraries provided by the LabVIEW Real-Time development environment.

If you use a stand-alone compiler and want to use cvi_lvrt.dll, include cvi\extlib\cvi_lvrt.lib in your external compiler project instead of cvirt.lib and cvisupp.lib. Remember that when you use an external compiler, you link to that compiler's ANSI C library.

If you use the **Create Distribution Kit** command on a project that you link for LabVIEW Real-Time Support only, LabWindows/CVI automatically includes cvi_lvrt.dll in the distribution kit and disables the option to distribute the full LabWindows/CVI Run-time Engine.

All the functionsin cvi_lvrt.dll are multithread safe.

- **Version Info**—When you click on this button, the Version Info dialog box appears. You can enter version information for the DLL in this dialog box. LabWindows/CVI saves the version information in the DLL as a standard Windows version resource. You can obtain the information from the DLL by using the Windows SDK functions `GetFileVersionInfo` and `GetFileVersionInfoSize`.

  In the Version Info dialog box, **File Version** and **Product Version** must be in the form:

  `n, n, n, n`

  where *n* is anumber from 0 to 255.

- **Import Library Choices**—This button lets you choose whether to create a DLL import library for each of the compatible external compilers or to create one only for the current compatible compiler. Refer to Chapter 3, *Compiler/Linker Issues,* in the *Measurement Studio LabWindows/CVI Programmer Reference Manual*. It also lets you choose to create the import libraries in the VXI*plug&play* subdirectories instead of the directory of the DLL.

  If you choose to use the DLL directory and create an import library for each compiler, LabWindows/CVI creates the files in subdirectories named `msvc` and `borland`. LabWindows/CVI also creates the library for the current compatible compiler in the directory of the DLL. If you choose to create an import library only for the current compiler, LabWindows/CVI creates the file in the directory of the DLL.

  If you choose to use the VXI*plug&play* directories and create an import library for each compiler, LabWindows/CVI creates the files in the subdirectories `msc` and `bc` under the VXI*plug&play* `lib` directory. If you choose to create an import library for the current compiler only, LabWindows/CVI creates the file in the appropriate subdirectory.

- **Type Library**—This button lets you choose whether to add a type library resource to your DLL. Also, you can choose to include links in the type library resource to a Windows help file. LabWindows/CVI generates the type library resource from a function panel (`.fp`) file. You must specify the name of the `.fp` file. You can generate a Windows help file from the `.fp` file by using the **Generate Windows Help** command in the **Options** menu of the Function Tree Editor window.

  This feature is useful if you intend for your DLL to be used from Visual Basic. For more information, refer to Chapter 3, *Compiler/Linker Issues,* in the *Measurement Studio LabWindows/CVI Programmer Reference Manual*.

- **LoadExternalModule Options**—The following options assist you in loading external modules.

  – **Enable LoadExternalModule**—Select this option if your project uses `LoadExternalModule`. This option is enabled by default. If your project does not use `LoadExternalModule`, disable this option to reduce the size of your DLL. If you disable the option but still use `LoadExternalModule`, LabWindows/CVI prompts you to enable full support. You must rebuild your project before the changes take effect.

  - **Add Files to DLL**—Use this button to select additional module files that you want to link into the DLL. These are modules that your project files do not directly reference but that are referenced by modules you load at run-time by calling `LoadExternalModule`.

  If you force a Windows SDK import library into your project, your DLL might not load. The Windows SDK import libraries contain functions that are not present on all versions of Windows. If the DLL on your system does not export all the functions in the import library, your DLL will not load. Instead of forcing an import library into your DLL, you can force only the functions you need into the DLL. To force specific functions into your DLL, create a table of function pointers, and add the functions to the table. For example, to force references to `CreateWindow` and `GetFreeDiskSpace`, you can add the following code to a source file in your project.

```
void* ReferenceFunctionsTable[] = {
CreateWindow,GetDiskFreeSpace,
}
```

- **Exports**—The following options assist you in exporting symbols.
  - **Export What**—This indicates your current method of choice for determining which symbols in the DLL to export to the users of the DLL. Use the **Change** button to change your choice.
  - **Change**—This button lets you select the method to use for determining which symbols in the DLL to export to the users of the DLL. You have the following choices:
    - **Include File Symbols**—You must name one or more include files that declare symbols defined globally in the DLL. The DLL exports the symbols you declare in the include files. You can select from a list of include files in the project.
    - **Symbols Marked for Export**—The DLL exports all symbols you define in the DLL with the qualifier `__declspec(dllexport)` or `export`.
    - **Include File and Marked Symbols**—The DLL exports all symbols you define in the DLL with the qualifier `__declspec(dllexport)` or `export` and the symbols you declare in the include files.
- **OK**—This button accepts the current inputs and closes the dialog box.
- **Cancel**—This button cancels the operation and removes the dialog box.

Note When you use the **Symbols Marked for Export** option or the **Include File and Marked Symbols** option and include in your project an object or library file that defines exported symbols, LabWindows/CVI cannot correctly create the import libraries for both of the compatible external compilers. This problem does not arise if you use only source code files in your DLL project.

For more information on creating DLLs, refer to Chapter 3, *Compiler/Linker Issues,* in the *Measurement Studio LabWindows/CVI Programmer Reference Manual*.

Target Settings for Static Libraries

When you set the **Target Type** to **Static Library** and select **Build»Target Settings**, the Target Settings dialog box has the following options:

- **Library File**—The name of the static library files for the debug and release versions of your program. Changing the value of the ring control allows you to edit the filename for the debug and release configurations. You can use the **Browse** button to select an existing filename.
- **Library Generation Choices**—This button lets you choose whether to create a static library for each of the compatible external compilers or create one for the current compatible compiler only. Refer to Chapter 3, *Compiler/Linker Issues,* in the *Measurement Studio LabWindows/CVI Programmer Reference Manual*. If you want to create a static library for both compilers, you must not include any object or library files in your project because such files are specific to a particular compiler.

  If you choose to create a static library for each compiler, LabWindows/CVI creates the files in subdirectories named `msvc` and `borland`. LabWindows/CVI also creates the library for the current compatible compiler in the parent directory.
- **OK**—This button accepts the current inputs and closes the dialog box.
- **Cancel**—This button cancels the operation and removes the dialog box.

Build»Compile File

You must compile your source code before you execute your project. Use the **Compile File** command to compile the selected source files. If LabWindows/CVI encounters any build errors, the Build Errors window appears with a list of errors.

Refer to the descriptions of in the *Options»Build Options* and *Options»Compiler Defines* sections.

Build»Mark File for Compilation

When LabWindows/CVI marks a source file for compilation, a *C* appears next to the filename in the Project window. LabWindows/CVI recompiles marked files the next time you build the project. When you modify a source file, LabWindows/CVI automatically marks the file for compilation. You can force LabWindows/CVI to compile a source file on the next build with the **Mark File for Compilation** command.

Build»Mark All for Compilation

Use the **Mark All Files for Compilation** command to force LabWindows/CVI to recompile all source files in the project the next time you build the project.

Build»External Compiler Support

Use the **External Compiler Support** command to help you build your executable or DLL in one of the two compatible external compilers. For more information on this topic, refer to Chapter 3, *Compiler/Linker Issues,* in the *Measurement Studio LabWindows/CVI Programmer Reference Manual*.

When you execute the command, the External Compiler Support dialog box appears.

The External Compiler Support dialog box has the following options:

- **UIR Callbacks**—This option creates an object or source file for you to link into your executable or DLL. The object or source file contains a list of the callback functions you specify in the User Interface Resource (`.uir`) files in your project. When you load a panel or menu bar from the `.uir` file, the User Interface Library uses the list to link the objects in the panel or menu bar to their callback functions in your executable or DLL. If you specify callback function names in your `.uir` file(s), set the ring control to Source File, enter the name of the source file to create, and click on the **Create** button. In the future, whenever you save modifications to any of the `.uir` files in the project, LabWindows/CVI automatically updates the source file.

  You must call the `InitCVIRTE` function at the beginning of your `main`, `WinMain`, or `DLLmain` function so that LabWindows/CVI run-time libraries can initialize the list of names from the source file. If you create a DLL and any of your callback functions are defined in but not exported by the DLL, you must call `LoadPanelEx` or `LoadMenuBarEx` (rather than `LoadPanel` or `LoadMenuBar`) from the DLL.
- **Using LoadExternalModule to Load Object and Static Library Files**—This option enables the section of the dialog box that you use when creating an executable or DLL that calls the Utility Library `LoadExternalModule` function to load object or static library files.

Note This option is not necessary if you use `LoadExternalModule` to load only DLLs that you load through DLL import libraries.

- Unlike DLLs, object and static libraries can contain unresolved external references. When you use `LoadExternalModule` to load an object or static library file, LabWindows/CVI resolves these references using symbols in your executable or DLL or in previously loaded external modules. Consequently, the names of the symbols in your executable or DLL that are necessary to resolve these references must be available to the `LoadExternalModule` function.

- **CVI Libraries**—This display provides information `LoadExternalModule` requires when your run-time modules reference symbols in any of the following LabWindows/CVI libraries:
  - User Interface Library
  - RS-232 Library
  - DDE Library
  - TCP Library
  - Formatting and I/O Library
  - Utility Library

  If you use one of these libraries, include in your external compiler project the source file displayed in this indicator. This does not apply if you use `LoadExternalModule` to load only DLLs.
- **ANSI C Library**—This display provides information `LoadExternalModule` requires when your run-time modules reference symbols in the ANSI C library. Include in your external compiler project the source file displayed in this indicator. This does not apply if you use `LoadExternalModule` to load only DLLs.
- **Other Symbols**—Select this option if your run-time modules refer to symbols other than those covered by the previous two options. Such symbols include functions or variables that you define globally in your executable or DLL and to which your object or static library run-time modules expect to link. This option creates an object file for you to link into your executable or DLL.
  - **Header File**—Insert the name of an include file that contains complete declarations of all the symbols necessary to resolve references from run-time modules.
  - **Object File**—Enter the name of the object file to create. Click on the **Create** button to create the file. You must include this file in your external compiler project.

The bottom of the External Compiler Support dialog box contains a list of library files for you to include in your external compiler project. The files are as follows:

- `cvi\extlib\cvirt.lib`
- `cvi\extlib\cvisupp.lib`
- `cvi\extlib\cviwmain.lib`

Use `cviwmain.lib` only when the external compiler requires you to define `WinMain`, when you do not define it in your project, and when any of the libraries the external compiler automatically links do not define it. In general, console applications do not require `WinMain`. GUI application wizards sometimes automatically include it in the source code they generate.

Build»Create Distribution Kit

Use the **Create Distribution Kit** command to make an installer application from which you can install your executable program on a target machine. **Create Distribution Kit** automatically includes all the files necessary to run your executable program on a target computer except for DLLs for National Instruments hardware, files that you load using `LoadExternalModule` and any ActiveX servers that are not the target of your project.

Do not include DLLs for National Instruments hardware in your distribution kit. Users can install the DLLs for their hardware from the distribution disks that they obtain from National Instruments.

If you load files using `LoadExternalModule`, you must include these files manually using the **Add/Edit Group** features of the **Create Distribution Kit** command. Refer to Chapter 4, *Creating and Distributing Stand-alone Executables and DLLs,* in the *Measurement Studio LabWindows/CVI Programmer Reference Manual*.

The following options are available in the **Create Distribution Kit** dialog box.

- The **Install Location** section of the Create Distribution Kit dialog box has the following options:
  - **Parent Folder**—The default directory on the target machine where your application will install. You can select from a list of common Windows locations. These locations automatically resolve to the actual directories on the target machine during the install.
  - **Sub Folder**—The subfolder that appears in the user installation. You can specify "\." to denote the same level as the Parent Folder.
- The **Build Information** section of the Create Distribution Kit dialog box has the following options:
  - **Build Location**—The path into which you want to build your distribution kit. At a minimum, you must have at least 3.5 MB of space in the build location to successfully create a basic distribution kit. Creating a distribution kit directly to a 1.44 MB floppy disk is not supported. To install your application on another machine, you must include all of the files from this directory and then launch `setup.exe`.
  - **Browse**—Browses for a build location.
  - **Installation Language**—The language that the installation program uses for text during the installation.

Note You may receive an error if you attempt to build a Japanese distribution kit on English Windows NT 4.0/9*x*. These operating systems do not have support for the Japanese codepage installed by default. You also can receive an error at distribution kit install time

on these same English versions of Windows. While there are several ways to install Japanese codepage support on your system, the following is a possible method:

- Install Microsoft Internet Explorer 5.0 or greater
- Go to `http://windowsupdate.microsoft.com` and go to the products update section.
- Download Japanese Language Support for Internet Explorer, and follow the installation intructions.
- Reboot, restart LabWindows/CVI, and retry the distribution kit build.

– **Run-Time Engine Support**—Selects which run-time engine is included in your distribution kit. You should match this choice to the run-time engine that your project is built with (in the Target Settings dialog). Making a selection other than **Full Runtime Engine** or **All** causes other options on the distribution kit to be unavailable because those options require **Full Runtime Engine** support to function. You can select the following options:

  - **Full Runtime Engine**—Includes the full LabWindows/CVI run-time engine (`cvirte.dll`) and its supporting files. If you choose **Install in Windows System Directory**, the run-time engine is installed into whatever the Windows System directory resolves to on the target machine. If you choose **Install in Application Directory**, the run-time engine is installed in the same directory as your application.
  - **Instrument Driver Only**—Includes support for instrument drivers. If you install instrument driver support, your project does not link to the entire set of LabWindows/CVI libraries but to a smaller set of functions. When you select this option, your distribution will not include the LabWindows/CVI run-time engine DLL, `cvirte.dll`. Instead, your distribution includes `instrsup.dll`, which is much smaller. This command allows other applications to use instrument driver DLLs without having to load the large LabWindows/CVI run-time engine DLL.

    If you choose **Install in Windows System Directory**, `instrsup.dll` is installed into whatever the Windows System directory resolves to on the target machine. If you choose **Install in Application Directory**, `instrsup.dll` is installed in the same directory as your application.

    Also, if you create a distribution kit for a project that you link for instrument driver support only in the Target Settings dialog box, LabWindows/CVI automatically enables the option to install `instrsup.dll`. You should not include `instrsup.dll` directly into a File Group—use this option instead.

    `instrsup.dll` contains functions from the following libraries:

    – Formatting and I/O Library (Except `ArrayToFile` and `FileToArray`)

    – RS-232 Library (Except `InstallComCallback`)

  - Utility Library (selected functions only; refer to the Utility Library Functions discussion later in this section)
  - ANSI C Library

  Your project also can link to the following libraries:

  - Analysis or Advanced Analysis Library
  - GPIB Library
  - VXI Library
  - VISA Library
  - IVI Library
  - Easy I/O for DAQ Library
  - Data Acquisition Library

  The following Utility Library functions are in `instrsup.dll`.

  - `Beep`
  - `DateStr`
  - `Delay`
  - `SyncWait`
  - `Timer`
  - `TimeStr`
  - `RoundRealToNearestInteger`
  - `TruncateRealNumber`
  - `InStandaloneExecutable`
  - `CVIRTEHasBeenDetached`

  `instrsup.dll` does not support the Standard Input/Output window. Functions such as `FmtOut` or `ScanIn` return errors when you use them with `instrsup.dll`. All the functions in `instrsup.dll` are multithread safe.

- **LabVIEW Real-Time Only**—If you select **LabVIEW Real-Time Only**, `cvi_lvrt.dll` is installed instead of `cvirte.dll` or `instrsup.dll`. Use this option for DLLs that will be downloaded and run on the LabVIEW Real-Time hardware.

  `cvi_lvrt.dll` is a subset of `instrsup.dll` and has the following additional restrictions:

  - The RS-232 Library is not available.
  - The file I/O and console I/O functions in the ANSI C, Formatting and I/O, and Utility libraries are not supported and will return run-time errors. Some of the functions affected by this are `printf`, `scanf`, `fprintf`, `fscanf`, `ScanIn`, `ScanFile`, `FmtOut`, `FmtFile`, `Beep`, `tmpfile`, and `tmpnam`.

– Only the “C” locale is recognized, and all case-sensitive functions such as `stricmp` and `tolower` perform conversions only from characters a–z to characters A–Z, and vice-versa, and only perform comparisons using this set of characters. Locale specific case-sensitivity is not honored.

– DLLs that link to `cvi_lvrt.dll` can be built only in the Release configuration, and you cannot debug them.

If you use a stand-alone compiler and want to use `cvi_lvrt.dll`, include `cvi\extlib\cvi_lvrt.lib` in your external compiler project instead of `cvirt.lib` and `cvisupp.lib`. Remember that when you use an external compiler, you link to that compiler’s ANSI C library.

If you use the **Create Distribution Kit** command on a project that you link for LabVIEW Real-Time Support only, LabWindows/CVI automatically includes `cvi_lvrt.dll` in the distribution kit and disables the option to distribute the full LabWindows/CVI run-time engine.

All the functions in `cvi_lvrt.dll` are multithread safe.

- **None**—Does not include any run-time engine support.
- **All Engines**—Includes support for all three of the above run-time engines.

Because it is not known at distribution kit build time if the target machine already has a run-time engine, NI recommends that you not choose **None** for your run-time engine support. Also, do not include run-time engine DLLs directly in a File Group, because the DLL might not run correctly or it might have uninstallation issues.

– **Run-Time Engine Install Location**—Selects the installation location for the run-time engine. If you choose **Install in Windows System Directory**, the run-time engine files are installed into whatever the Windows `system` directory resolves to on the target machine. If you choose **Install in Application Directory**, all run-time engine and supporting files, including `cvirt.dll` and `cvirte.dll`, are installed in the same directory as your application. The low-level support driver files cannot be installed in the same directory as your application.

– **Install DataSocket Support**—Set this control to **If Needed** to include the DataSocket library files in your distribution only if your project uses the DataSocket library. Set this control to **Always** to include the DataSocket files in your distribution regardless of whether your project uses the DataSocket library. Set this control to **Never** if you do not want to include DataSocket files in your distribution. DataSocket requires Internet Explorer 5.0 or later installed on the target machine to execute properly.

– **Install NI Reports Support**—Set this control to **If Needed** to include the NI Reports instrument driver in your distribution only if your project uses the NI Reports instrument driver. Set this control to **Always** to include the NI Reports files in your distribution regardless of whether your project uses the NI Reports

instrument driver. Set this control to **Never** if you do not want to include NI Reports files in your distribution.

– **Install Low-Level Support Driver**—This option lets you choose whether to install the LabWindows/CVI low-level support driver on the user's computer. The Utility Library functions shown in the following table require the LabWindows/CVI low-level driver. If you use any of these functions in your application, you must enable this option.

Function	Platforms that Require the Low-level Support Driver
`inp`	Windows 2000/NT
`inpw`	Windows 2000/NT
`outp`	Windows 2000/NT
`outpw`	Windows 2000/NT
`ReadFromPhysicalMemory`	All
`ReadFromPhysicalMemoryEx`	All
`WriteToPhysicalMemory`	All
`WriteToPhysicalMemoryEx`	All
`MapPhysicalMemory`	All
`UnMapPhysicalMemory`	All
`DisableInterrupts`	Windows 98/95
`EnableInterrupts`	Windows 98/95
`DisableTaskSwitching`	Windows 98/95

The LabWindows/CVI run-time engine loads the low-level support driver automatically at startup if the support driver is present on disk.

– **Install ActiveX Container Support**—Check this box to install ActiveX container support on the user's computer. If your project creates or includes any ActiveX servers (such as ActiveX controls), you must enable this option. You also must enable **Register Files as ActiveX Servers** if you distribute ActiveX controls in your project.

– **Install Win95 DCOM/RTE Support**—Enable this option to install DCOM support for the run-time engine. If you are installing to Windows 95 without Internet Explorer 4.0 or greater, then you must enable this option. If enabled, the distribution kit will correctly install DCOM only on those systems running Windows 95.

- The **File Groups** section of the Create Distribution Kit dialog box has the following options:
  - **File Groups**—Separates the files in your distribution kit into groups. You must assign a destination directory to each group. The distribution kit creates the directories on the target machine and places each of the file groups in its assigned directory. You can set each of the options to the right of the list box to different values for each file group. File groups for the files in your project are automatically created for you.
  - **Add Group**—Adds a new group to your distribution kit.
  - **Edit Group**—Edits the selected group.
  - **Delete Group**—Deletes the selected group from your distribution kit.
  - **Create Shortcuts**—Use this option to choose whether to create shortcuts that contain icons for files in the selected file group. The installation program will install the embedded icons for `.exe` files and the default icons for `.com`, `.pif`, `.lnk`, `.bat`, `.cmd`, `.hlp`, `.txt`, `.doc`, `.wri`, `.xls`, `.ppt`, `.htm`, `.html`, `.vsd`, `.xml`, `.pdf` and `.reg` files.

    To create a shortcut to the same file in more that one location, you can add the file to a new file group with the same Group Destination and change the shortcut in that new group to be created in the additional location. Items created on the Start menu are shortcuts, so they do not have file extensions. If you create shortcuts for two files of the same base name, such as `test.exe` and `test.hlp`, two shortcuts with the name "`test`" are created, and one overwrites the other. Make sure that you name shortcuts created in the same location differently.

    You can create shortcuts choose in the following locations:

    - **Start Programs (+Sub-Folder)**—LabWindows/CVI creates shortcuts in **Start»Program Files»Sub-Folder»shortcut** name. You can change the Sub-folder in the Advanced Distribution Kit Options dialog box. The default for the Sub-Folder is usually the name of the current LabWindows/CVI project.
    - **Start Programs (Top-Level)**—LabWindows/CVI creates shortcuts in **Start»Program Files»shortcut** name.
    - **Start (Top-Level)**—LabWindows/CVI creates shortcuts in the top portion of the menu directly under the **Start** button.
    - **None**—No shortcuts are created for the files in this group.
    - **Windows Startup**—LabWindows/CVI creates shorcuts in the Windows Startup folder. The items in the Windows Startup folder execute automatically every time a user logs into the system.

- **Desktop**—LabWindows/CVI creates shortcuts on the user's Windows desktop.
- **Send To Menu**—LabWindows/CVI creates shortcuts in the user's Send To menu. The Send To menu appears when you right-click a file in Windows Explorer.

– **Group Destination**—Sets the root destination directory for the selected group. You can select the destination directory to be either the application install location or a list of common Windows locations. These locations automatically resolve to the actual directories on the target machine during the install.

– **Relative Path**—Assigns a relative path based on the group destination directory in which to install the selected file group.

Note Using the "`..\`" syntax can cause unexpected results if you attempt to reference a directory that is "higher" than one of the common Windows locations. For example, if you set the **Group Destination** to **Windows System Directory** and the **Relative Path** to `..\Test`, you expect the Group Destination to resolve to `c:\WINDOWS\system`, and the relative path to resolve to `c:\WINDOWS\test`. However, due to the behavior of the installer, the relative path resolves to `c:\test` because the common Windows locations behave as if they were one single directory, no matter how many elements are in the path. Set the **Group Destination** to **Windows Directory**, and set the Relative Path to Test to obtain the expected location.

– **Distribute Objects/Libraries For Both Compilers**—This option helps you distribute object files, static libraries, and DLL import libraries for all the compatible external compilers. When enabled, this option affects all the `.obj` and `.lib` files listed in the selected file group. LabWindows/CVI includes two versions of each file in the distribution kit. LabWindows/CVI expects these versions to be in subdirectories under the specified location of each file. The subdirectories must be named `msvc` or `borland`. For example, if you specify the file `c:\myapp\distr\big.lib` in a file group and you enable the **Distribute Objects/Libraries For Both Compilers** option, when LabWindows/CVI creates the distribution kit, you must have the following files on your disk:

```
c:\myapp\distr\msvc\big.lib
```

```
c:\myapp\distr\borland\big.lib
```

The installation program prompts the user to choose one of the compatible external compilers. The installation program installs only the files for the compiler the user chooses at distribution kit install time.

You might want to use this feature if you distribute modules for use with the LabWindows/CVI development environment or external compilers. If you distribute a turnkey application, this feature is not necessary.

Note Do *not* use this feature for distributing DLL import libraries for VXI*plug&play* instrument drivers. When installing a VXI*plug&play* instrument driver, you must install two import libraries: one compatible with Visual C/C++ and the other with Borland C/C++. The two import libraries must be installed in the `msc` and `bc` subdirectories under the VXI*plug&play* `lib` directory. LabWindows/CVI sets this up automatically for you if you use the **Create DLL Project** command in the Function Tree Editor window with the **VXIplug&play Style** command enabled. For more information, refer to the *Distributing VXIplug&play Instrument Drivers* topic in the **LabWindows/CVI Help**.

  – **Register Files As ActiveX Servers**—Enable this option if you distribute any ActiveX servers (such as ActiveX controls) in your project. If you use an ActiveX control in your project, you must manually include the control in the File Groups list box and enable the **Register Files As ActiveX Servers** and **Install ActiveX Container Support** options. Enabling **Register Files As ActiveX Servers** causes LabWindows/CVI to attempt to register all the files in the file group, regardless of file type, as ActiveX servers. The installation does not show any warnings or errors for files that failed to successfully register as ActiveX servers.

    If a project is an ActiveX server, the distribution kit registers the project as an ActiveX server, regardless of whether you enable this option.

- **Advanced**—Opens the Advanced Distribution Kit Options dialog box.
- **Default**—Resets all controls in the Create Distribution Kit dialog box to their default values. When you create a new distribution kit, you can click **Default** to undo changes you have made to the controls in the dialog box.

Note When you use the Create Distribution Kit dialog box to modify an existing distribution kit, **Default** replaces your existing file groupings and settings with default values. If you click **Default** in error, click **Cancel** in the dialog box that appears to prevent this change to your distribution kit.

- **Build**—Builds your distribution kit.
- **Cancel**—Cancels the **Create Distribution Kit** operation.

Refer to the *Customizing Create Distribution Kit Installers in LabWindows/CVI 6.0* application note for information on how you can customize your Distribution Kit.

Run Menu

This section explains how to use the commands in the Project window **Run** menu, as shown in Figure Run Menu. You can use commands in the **Run** menu to run your program and assign breakpoints. For more information about breakpoints, refer to the *Introduction to Breakpoints and Watch Expressions* section of Chapter 5, *Source and Interactive Execution Windows*.

Run»Debug

If the **Target Type** is **Executable**, the **Debug** menu item runs the project's target executable for the currently selected configuration. You set the active configuration using the **Configuration** submenu in the **Build** menu. If the **Target Type** is **Dynamic Link Library**, the **Debug** menu item runs the executable you specify with the **External Process** menu item. Before running the executable, the **Debug** menu item compiles any source files that must be compiled and builds the project's target executable or DLL if you made changes since you last built the target DLL or executable.

Run-Time Error Reporting

During the execution of a program, LabWindows/CVI can report various run-time errors. One example of a run-time error is a call to a LabWindows/CVI library function in which an array or string argument is not large enough to hold the output data.

When such errors occur, a dialog box appears, identifying the type of error and the location in the program where the error occurred. LabWindows/CVI lists the error in the Run-Time Errors window.

LabWindows/CVI then suspends the program so you can inspect the values of variables in the Variables window. To terminate a program that has been suspended because of a run-time error, select the **Terminate Execution** command or press <Ctrl-F12> while a LabWindows/CVI Environment window is active.

Debugging DLLs

If you set the **Target Type** item in the **Build** menu to **Dynamic Link Library** and set the **Configuration** item in the **Build** menu to **Debug**, the **Create Debuggable Dynamic Link Library** command appears in the **Build** menu. When you use the **Create Debuggable Dynamic Link Library** command, LabWindows/CVI includes debug code in your DLL and generates an extra file that contains a symbol table and source position information necessary for debugging. The extra file has the same pathname as the DLL except that its extension is `.cdb`.

In the LabWindows/CVI development environment, you can debug only DLLs you create in LabWindows/CVI with the **Create Debuggable Dynamic Link Library** command. Other development environments cannot debug DLLs you create in LabWindows/CVI.

The amount of debugging information included in the DLL and debug file depends on the value of the **Debugging Level** control in the Build Options dialog box.

Location of Files Required for Debugging DLLs

To debug a DLL in LabWindows/CVI, the `.cdb` file and the source files for the DLL must be available. LabWindows/CVI looks for the `.cdb` file in the following locations and order:

1. The directory from which LabWindows/CVI loaded the DLL
2. The directory in which you created the DLL
3. The directory of the current project target, if the current project target is the DLL
4. The Windows directory
5. The Windows `system` directory

If LabWindows/CVI cannot find the `.cdb` file in any of these locations, a dialog box prompts you to browse for it. After you enter the location of the `.cdb` file, LabWindows/CVI stores the location in the Windows Registry.

The `.cdb` file contains the locations of the source files at the time you created the DLL. It also contains the LabWindows/CVI installation directory and VXI*plug&play* framework directory. When LabWindows/CVI has to display a DLL source file, it looks for the file in the following places and order:

1. The project list, if the current project target is the DLL you are debugging
2. The source file directory that LabWindows/CVI stored in the `.cdb` file
3. If you have moved the `.cdb` file, the directory that is in the same relative position to the current `.cdb` location as the stored source file directory was in relation to the original `.cdb` file location
4. If the source file was originally under the LabWindows/CVI directory and the LabWindows/CVI directory has changed, the same relative position to the new LabWindows/CVI directory
5. If the source file was originally under the VXI*plug&play* framework directory and the VXI*plug&play* framework directory has changed, the same relative position to the new VXI*plug&play* framework directory

If LabWindows/CVI cannot find a DLL source file, it reports an error. Make sure that your files are in one of the preceding locations.

In summary, when you move a DLL to another machine and you want to debug it there, you must also copy the `.cdb` file and the source files. It is best to keep the `.cdb` file and source files in the same relative location to each other. You do not have to keep the `.cdb` file in the same directory as the DLL. LabWindows/CVI prompts you for the `.cdb` file location and keeps track of it thereafter.

Different Ways to Debug DLLs

You can debug a DLL two ways. In one approach, you run a LabWindows/CVI executable project that calls the DLL. The DLL project is not open in LabWindows/CVI. In the other approach, you open the DLL project and run an external process that uses the DLL.

Running a Program in LabWindows/CVI

To debug a DLL that another LabWindows/CVI project uses, select **Run»Debug** after loading the project that uses the DLL. When LabWindows/CVI loads the DLL, it loads the corresponding debug information from the `.cdb` file. LabWindows/CVI honors breakpoints you set in DLL source files. LabWindows/CVI saves in the project any breakpoints you set in any source file, regardless of whether the source file is in the project.

Also, you can set watch expressions for a debuggable DLL. For each watch expression, you must choose whether it applies to a project or a DLL. If it applies to a DLL, you must enter the name of the DLL. LabWindows/CVI stores this information in the project. For more information, refer to Chapter 10, *Variables and Watch Windows*. You can debug multiple DLLs called from the same project. LabWindows/CVI handles each DLL in the same manner.

Running an External Process

To debug a DLL that an external process uses, load the DLL project into LabWindows/CVI. Select **Run»Select External Process** in the Project window. A dialog box appears that allows you to enter the pathname of an external process and command-line arguments. LabWindows/CVI stores this information in the project.

After you specify the pathname of an external process, the **Debug Project** item in the **Run** menu changes to **`Debugxxx.exe`**, where ***`xxx.exe`*** is the filename of the external process. When you select the **Debug xxx.exe** command, LabWindows/CVI starts the external process and attaches to it for debugging. If you have set any breakpoints in the source files for the DLL, LabWindows/CVI honors them.

If the external process loads other debuggable DLLs, you can debug them even though a project for a different DLL is open. LabWindows/CVI handles the other DLLs as described in the previous section, *Running a Program in LabWindows/CVI*.

The command-line arguments that you set in the Select External Process dialog box are the same command-line arguments that you set using **Options»Command Line** in the Project window.

Run»Continue

Use the **Continue** command to resume program execution when in a breakpoint state.

Run»Terminate Execution

Use the **Terminate Execution** command to terminate a program that is in a breakpoint state.

Run»Break at First Statement

Select **Break at First Statement** to force LabWindows/CVI to break a program on the first executable statement. When activated, this command has a checkmark beside it in the menu.

Run»Breakpoints

The **Breakpoints** command opens the Breakpoints dialog box, which contains a list of the breakpoints in the project. Also, you can open this dialog box by right-clicking in the line icons column and selecting **Breakpoints** from the pop-up menu.

Use the **Add/Edit Item** button to edit a single breakpoint with the Edit Breakpoint dialog box. The Edit Breakpoint dialog box contains the following items:

- **File**—Use this control to select the source file that contains the breakpoint you want to edit.
- **Line**—Use this control to select the line that contains the breakpoint you want to edit.
- **Pass Count**—Use this control to select the number of times that the source code line executes before the breakpoint occurs.
- **Condition**—Use this box to enter an optional expression that LabWindows/CVI evaluates before it executes the source code line. If the condition is true, your program enters a breakpoint state; otherwise, execution continues. Refer to the *Conditional Breakpoints* section in Chapter 5, *Source and Interactive Execution Windows*, for examples of conditional expressions.
- **Disabled**—Use this checkbox to disable the breakpoint. The breakpoint icon in the Source window changes color to indicate that you disabled it.
- After you set all the breakpoint attributes in the Edit Breakpoint dialog box, you can **Replace** the breakpoint with the new attributes, **Add** the breakpoint to the breakpoint list, or **Cancel** the operation.

The **Go to Line** button takes you to the source code location of the currently selected breakpoint.

The **Delete Item** button deletes the currently selected breakpoint.

The **Delete All** button deletes all the breakpoints.

The **Disable All** button forces LabWindows/CVI to ignore all the breakpoints. The breakpoint icons in the Source window change color to indicate that you disabled them.

The **Enable All** button activates all the breakpoints. The breakpoint icons in the Source window change color to indicate that you enabled them.

The **OK** button accepts the current breakpoint attributes, and the **Cancel** button cancels the current operation.

Run»Select External Process

This command applies only when you set the **Target Type** item in the **Build** menu to **Dynamic Link Library**. Use the **Select External Process** command to specify a stand-alone executable that uses your DLL. When you execute the command, a dialog box appears in which you enter the pathname and command-line arguments to an external program. The **Run Project** item in the **Run** menu then changes to **Debug *xxx*.exe**, where ***xxx*.exe** is the filename of the external program. When you execute the **Debug xxx.exe** command, LabWindows/CVI starts the external process and attaches to it for debugging. If you have set any breakpoints in the source files for the DLL, LabWindows/CVI honors them.

LabWindows/CVI stores external program pathname and command-line arguments in the project.

Run»Execute

The **Execute** command launches the executable for the active configuration without attaching the debugger to the executable. You must create the executable, using the **Create** menu item in the **Build** menu, before you use this command. This command is dimmed if the **Target Type** for the project is DLL or Static Library.

Run»Threads

The **Threads** command brings up a dialog box listing the threads in the program being debugged. Use this dialog box to select the threads whose local variables and call stack you wish to view. When you select a thread from this dialog box and click on **OK** to close the dialog box, LabWindows/CVI displays the local variables for the selected thread in the variable display and displays the current source position of the thread in a Source window. The **Up Call Stack**, **Down Call Stack**, and **Call Trace** commands in the Source windows **Run** menu display information on the currently selected thread. The Watch display shows the thread specific values of the expressions in the Watch window.

Using Instrument Drivers

This section presents a general overview of instrument drivers. Refer to the *Measurement Studio LabWindows/CVI Instrument Driver Developers Guide* for more information on creating instrument drivers.

An instrument driver is a set of high-level functions with graphical function panels that make programming easier. It encapsulates many low-level operations, such as data formatting and communication with GPIB, RS-232, and VXI, into intuitive, high-level functions. An instrument driver usually controls a physical instrument, but it also can be a software utility.

Instrument driver programs have an associated include file that declares the high-level functions you can call, the global variables you can access, and the defined constants you can use.

Instrument Driver Files

A LabWindows/CVI instrument driver typically consists of the following three or four files. Each file has the same base filename, which is an abbreviation of the actual instrument name. The instrument driver files must reside in the same directory on your disk, or they must be in the appropriate VXI*plug&play* directories.

- The function panels are in a file with the extension `.fp`. Refer to Chapter 6, *Using Function Panels,* for a detailed description of function panels.
- For instrument drivers that use an attribute model, such as IVI drivers, there can be an additional `.sub` file that contains attribute information displayed in the function panels.
- The function, variable, and defined constant declarations are in an include file with a `.h` extension.
- The instrument driver program can be in one of several different types of files.
  - A source file with a `.c` extension.
  - An object file that contains one or more compiled C modules with a `.obj` extension or a library file with a `.lib` extension. The compilation must be done by LabWindows/CVI or a compatible external compiler. Refer to Chapter 3, *Compiler/Linker Issues,* in the *Measurement Studio LabWindows/CVI Programmer Reference Manual* for more information on compatible external compilers.

For example, the instrument module files for a Fluke 8840A multimeter are `fl8840a.fp`, `fl8840a.c`, and `fl8840a.h`.

You can load an instrument driver into the LabWindows/CVI interactive program whether the instrument program is in the form of a `.c`, `.obj`, or `.lib` file. The presence of the `.h` file is essential because you must include it in your program to reference functions, global variables, and constants in the instrument driver.

VXI*plug&play* Instrument Driver Files

When you install a VXI*plug&play* instrument driver, the installation program does not place the include (`.h`) file in the same directory as the `.fp` file. The installation program places it in the `include` subdirectory under the VXI*plug&play* directory. LabWindows/CVI can find the include files in the VXI*plug&play* include directory.

When you install a VXI*plug&play* instrument driver, the installation program places the source (`.c`) file in the same directory as the `.fp` file. The installation program also installs a dynamic link library (`.dll`) and two import libraries (`.lib`), one compatible with Visual C/C++ and the other with Borland C/C++. These files are not in the same directory as the `.fp` file. The import libraries are in the `msc` and `bc` subdirectories in the VXI*plug&play* `lib` directory. The DLL is in the VXI*plug&play* `bin` directory. The installation program adds the VXI*plug&play* `bin` directory to the `PATH` environment variable so that the DLL can be found using the standard Windows DLL search algorithm.

If the `.fp` file is under the VXI*plug&play* framework directory and your current compatible compiler is Visual C/C++ or Borland C/C++, LabWindows/CVI can find the appropriate import library. If LabWindows/CVI finds the import library, it gives it precedence over the `.c` file as the program file for the instrument driver.

Loading/Unloading Instrument Drivers

You can load and unload instrument drivers manually using the **Instrument** menu. Instrument drivers loaded through the **Instrument** menu do not have to be listed in the project, and you can load or unload them at any time except during program execution.

You can incorporate instrument drivers into the project by selecting **File»Add to Project** in a Function Panel window or the Function Tree Editor window or by selecting **Edit»Add Files to Project** in the Project window. The `.fp` file represents the instrument driver in the project list. If the `.fp` file is in the project list when you open the project, LabWindows/CVI automatically loads the instrument driver and removes it when you unload the project.

Precedence Rules for Loading the Instrument Driver Program File

When you load a `.fp` file, LabWindows/CVI loads the instrument driver program file. In some cases, you might have an instrument driver program file in more than one format. For instance, you might have `fl8840a.obj` and `fl8840a.c` in the same directory. This can occur when you obtain the source code for the instrument driver and then compile it. LabWindows/CVI chooses which file to load according to the following rules:

- If an instrument driver program file is in the project, LabWindows/CVI loads it. There can be at most one unexcluded program file with the same base name as the `.fp` file in the project list. Thus, *`x.obj`* and *`x.c`* cannot be in the project list at the same time unless you exclude one or both of them.

- If both of the following conditions apply, the `.lib` file is associated with the `.fp` file:
  - The `.fp` file is under the VXI*plug&play* framework directory.
  - A `.lib` file is in the appropriate VXI*plug&play* framework subdirectory. The following table, VXI*plug&play* Framework Subdirectories, shows the corresponding subdirectories.

Compatible Compiler	Corresponding Subdirectory that Contains the .lib File
Visual C/C++	`lib\msc`
Borland C/C++/C++ Builder	`lib\bc`

- If an instrument driver program file is on disk in the same directory as the `.fp` file, LabWindows/CVI loads it with the following precedence:
  1. `.lib`
  2. `.obj`

Loading an Instrument without an Instrument Program

You can load a `.fp` file as an instrument, even if no program file exists for it. In this case, LabWindows/CVI does not associate a program with the `.fp` file. Nevertheless, the `.fp` file appears in the **Instrument** menu.

This is useful if you want to use `.fp` files for documenting functions in your project. When you do not provide a program file for the `.fp` file, you cannot execute the function panels, but you can insert code into Source windows from them.

If you try to execute an instrument driver function panel when no program is associated with the instrument, LabWindows/CVI reports a run-time error. It is possible to associate a `.c` file with a `.fp` file after you load the `.fp` file. Refer to the *Instrument»Edit* command for more information.

Modules that Contain Non-Instrument Functions

Although the LabWindows/CVI instrument driver mechanism is primarily for program modules that control instruments, you can use it for any module that contains a set of high-level functions.

Suppose, for instance, you write a set of specialized analysis functions. If you develop function panels and a `.h` file for the module, you can load the module from the **Instrument** menu and call the functions from the function panels.

Modifying an Instrument Driver

You might want to modify an instrument driver that you received from National Instruments or elsewhere. If you want to modify the instrument driver program file, you must have the `.c` file for the instrument driver.

Before modifying an instrument driver, familiarize yourself with the *Measurement Studio LabWindows/CVI Instrument Driver Developers Guide*.

You can modify four parts of an instrument driver:

- You can modify the function tree by selecting the `.fp` file using the **File»Open»Function Tree (.fp)** command, by selecting **Instrument»Edit**, or by selecting **Tools»Edit Function Tree** from a Source window that contains the instrument driver source or include file.
- You can modify the function panels by selecting **Option»Edit Function Panel Window** from a Function Panel window, by selecting **Edit»Edit Function Panel Window** from a Function Tree Editor window, or by selecting **Tools»Edit Function Panel** from a Source window that contains the instrument driver source or include file when the text cursor is over the name of the function in the driver.
- You can modify the instrument driver program file by selecting **Instrument»Edit** in a Function Panel window, by selecting **Tools»Go To Definition** from a Function Panel Editor window, or by selecting **Go To Definition** from the context menu in the Function Tree Editor window.
- You can modify the instrument driver include file by selecting the `.h` file using the **File»Open»Include (*.h)** in the Project window, by selecting **Tools»Go To Declaration** from a Function Panel Editor window, or by selecting **Go To Declaration** from the context menu of the Function Tree Editor window.

Instrument Driver Fundamentals

If you want to add functions to a driver, or if you develop a driver without using the Instrument Driver Development Wizard, you must know the instrument driver fundamentals in this section. Although the wizard is the preferred method for developing a driver for an instrument that you control through a GPIB, VXI, or serial interface, many users develop instrument drivers for other purposes, such as common utility functions, analysis algorithms, or for controlling custom hardware that does not fit the IVI instrument driver model for GPIB or VXI devices.

Defining the Instrument Functions

An instrument driver exports a set of functions that programmers can use to control an instrument. To make the set of functions easy to use, you must organize them into a logical structure. Group related functions into categories or classes. For complex instruments, group related classes into higher-level classes. For very complex instruments that incorporate multiple personalities, you might consider creating multiple instrument drivers.

Structuring Functions in an Instrument Driver

If you use the wizard with a predefined instrument template, the wizard creates a function hierarchy for you. Otherwise, you must define and structure the driver functions on your own.

The three implementations of a single instrument driver hierarchy in this section show you some options for structuring functions. In this example, the driver includes seven functions with which to program the instrument.

The first implementation gives the user a simple linear list of all available functions.

```
instrumentA(1)
    function1
    function2
    function3
    function4
    function5
    function6
    function7
```

The second implementation breaks the functions into two function classes.

```
instrumentA(2)
    function_class1
        function1.1
        function1.2
        function1.3
        function1.4
    function_class2
        function2.5
        function2.6
        function2.7
```

The third implementation treats the two function classes as two distinct instruments.

```
instrumentA(3.1)
    function1
    function2
```

```
    function3
    function4
instrumentA(3.2)
    function5
    function6
    function7
```

To successfully structure the functions for your instrument, you must determine who will use the instrument driver and how they will use the instrument. Define functions that stand alone to perform a useful action. For example, it might at first seem logical to use the functions `SetDMMRange` and `SetDMMFunction` for setting the range and function of a multimeter. However, a more useful function might be `ConfigureMeasurement`, for setting up multiple parameters.

Defining the Hierarchy of Functions

It is important to design the function hierarchy for the instrument driver carefully. When you do, the user can identify the functions required by the desired action without the burden of choosing from a long list of unrelated functions.

The concept of function classes is only apparent to the user from within the LabWindows/CVI development environment. The application program calls all functions within an instrument driver the same way, regardless of which function class they are in.

Defining the Function Parameters

To design the code for an instrument driver function, you must first establish its parameters.

Function parameters provide input information to the function and output variables where the function can store its results. Output parameters often contain values that the function reads from the instrument and formats for the user.

Data Types

You must define a data type for each parameter in each instrument driver function. All data types the instrument driver uses must be intrinsic C data types or data types that you define in the `.h` file and list in the `.fp` file. You specify the data type of a parameter when you create its corresponding control on a function panel. This data type must also be consistent with the function prototypes in the instrument driver header file.

LabWindows/CVI uses the data type information to implement the variable declaration and run-time checking capabilities when users operate function panels. When you declare a variable from a function panel, LabWindows/CVI presents options based on the data type you

specify for the function panel control. When you run a function from a function panel, LabWindows/CVI verifies that the data type of the control matches the prototype of the function.

Data types are divided into three classes: predefined data types, user-defined data types, and VISA data types.

Predefined Data Types

Predefined data types are available by default in the LabWindows/CVI environment. The predefined data types consist of intrinsic C data types and meta data types that LabWindows/CVI defines.

Intrinsic C Data Types

The intrinsic C data types that LabWindows/CVI defines are as follows:

```
int
long
short
char
unsigned int
unsigned long
unsigned short
unsigned char
int []
long []
short []
char []
unsigned int []
unsigned long []
unsigned short []
unsigned char []
double
float
double []
float []
char *
char *[]
void *
```

When you create a control to represent an array of data, make the data type an intrinsic C data type that ends with the square brackets, `[]`. Do *not* select a data type that ends with an asterisk `(*)`. The brackets tell LabWindows/CVI that the control represents an array of data, not a

pointer. LabWindows/CVI can then perform the appropriate variable declaration and run-time checking capabilities when the user operates the function panel.

When you define a function panel control with an intrinsic C data type, variables the user declares in the control through the **Declare Variable** command appear with that data type in the dialog box. You must define the parameter with the same data type when you prototype the function in the instrument driver include file.

Meta Data Types

The meta data types are useful for parameters through which users can pass arguments of more than one data type. The meta data types are `Numeric Array`, `Any Array`, `Any Type`, and `Var Args`. Each of these data types defines a set of multiple allowable data types. When the user executes the **Declare Variable** command on a control defined with a meta data type, the user can select from a list of the allowable data types.

Numeric Array

Numeric Array specifies a parameter that can be any of the intrinsic C numeric array data types. You must define the parameter as `void *` in the function prototype. An example of a Numeric Array data type is in the `PlotX` function of the User Interface library. The `PlotX` function plots the values of any intrinsic C numeric array data type to a graph control on a user interface panel. On the function panel, the X Array control is of type `Numeric Array`. X Array is defined as `void *` in the following function prototype.

```
int PlotX (int panel, int control, void *xArray, int numPoints, int
           xDType, int plotStyle, int pointStyle, int lineStyle, int
           pointFreq, int color);
```

Any Array

Any Array specifies a parameter that can be any of the intrinsic C or user-defined array data types. You must define the parameter as `void *` in the function prototype. An example of an `Any Array` data type is in the `memcpy` function of the ANSI C library. This function copies a specified number of bytes from a target buffer of any type to a source buffer. In the function panel, the first parameter is the Target Buffer, which is of type `Any Type`. The Target Buffer is defined as `void *` in the function prototype.

```
void *memcpy(void *, const void *, size_t);
```

Any Type

Any Type specifies a parameter that can be any of the intrinsic C or user-defined data types. If the parameter is an output parameter, you must define it as `void *` in the function prototype. If the parameter is an input parameter, you must define it as "`...`" in the function prototype, and it must be the last parameter in the function. The Value output parameter of the

GetCtrlAttribute function in the User Interface Library is an example of the Any Type data type. The function obtains the value of a particular attribute for a particular user interface control. Different attributes have different data types. Users pass the attribute value parameter by reference, and it is therefore a pointer. Consequently, the attribute value parameter is of type Any Type and is defined as void * in the following function prototype shown below.

```
int GetCtrlAttribute(int panel, int control, int attribute, void
    *value);
```

The Value parameter of the SetCtrlAttribute function also applies to attributes of different data types, but it is an input rather than an output parameter. Users pass this parameter by value rather than by reference, and thus it can have different sizes. For example, it might be an int or a double. Consequently, the attribute value parameter is of type Any Type and is defined as "..." in the following function prototype.

```
int SetCtrlAttribute (int panel, int control, int attribute, ...);
```

Var Args

Var Args specifies a variable number of parameters that can be any of the intrinsic C or user-defined data types. You must define the parameters as "..." in the function prototype. The printf and scanf functions in the ANSI C library have examples of the Var Args data type. Following the format string parameter in each function, you can specify one or more parameters of different data types to match the type specifiers in the format string. In printf, the parameters are passed by value. In scanf, they are passed by reference and thus are really pointers. For both functions, one Var Arg function panel control is used, and "..." appears in the following function prototypes.

```
int printf (const char *, ...);
int scanf (const char *, ...);
```

User-Defined Data Types

LabWindows/CVI also lets you define data types and use them in function panels. You must declare user-defined data types in the function panel file of an instrument driver, and you must define the data type in the include file for the driver. Declare user-defined data types with the Data Types command box in the Function Panel Editor.

For example, you can define a waveform_var data type for an instrument driver to represent waveform data. This waveform_var data type could be a structure that contains an array of doubles to represent the individual points in the waveform, a float for the time of the first point, and a float for the time between points.

Creating a User-Defined Data Type

Complete the following steps to create a user-defined data type for use in a function panel.

1. Define the data type with a `typedef` statement in the instrument driver header file.
2. Add the data type to the instrument driver function panel file using the **Options»Data Types** command in the Function Panel Editor.
3. Using the previous example of the `waveform_var` data type, include the following code in the include file for the instrument driver.

   ```
   typedef struct {double waveform_arr [500];float t_zero;float
      t_delta;} waveform_var;
   ```

4. Make the `waveform_var` data type available in the function panel file. Select **Options»Data Types** in the Function Panel Editor and enter `waveform_var` in the Type box of the Edit Data Type List dialog box. Then click on **Add**.

Now you can select the `waveform_var` data type when you create function panel controls for this instrument driver. Also, users can interactively declare a variable of `waveform_var` data type from any function panel control that you define as `waveform_var`.

User-Defined Array Data Types

Use care when you declare user-defined data types that are arrays. If you want to define a user-defined array data type, square brackets `[]` must appear at the end of the type name in the Edit Data Type List dialog box. The brackets enable the interactive variable declaration and other capabilities of LabWindows/CVI function panels. For example, to declare an array of `waveform_var` type from the example presented above, add

`waveform_var []` (This example is correct because it includes brackets.)

in the **Type** box of the Edit Data Type List dialog box and include the `typedef` declaration for `waveform_var` in the driver include file.

Examples of incorrect ways to define user-defined array data types are shown below.

Assume the following data type definitions are in an instrument driver header file.

```
typedef waveform_var * waveform_arr1;
typedef waveform_var waveform_arr2[100];
```

Then the following data type declarations in the Edit Data Type List dialog box are incorrect:

`waveform_var *` (This example is incorrect because it lacks brackets.)

`waveform_arr1` (This example is incorrect because it lacks brackets.)

`waveform_arr2` (This example is incorrect because it lacks brackets.)

VISA Data Types

The VISA I/O library defines a special set of data types. The IVI Library also uses some of these data types. The data types strictly define the type and size of the parameters and therefore promote the portability of the functions to new operating systems and programming languages.

A subset of the VISA data types has been defined for use in the development of LabWindows/CVI instrument drivers and are accessible as user-defined data types. The following table shows these special data types for instrument drivers.

VISA Type Name	Definition
`ViInt16`	Signed 16-bit integer
`ViInt32`	Signed 32-bit integer
`ViUInt16`	Unsigned 16-bit integer
`ViUInt32`	Unsigned 32-bit integer
`ViReal64`	64-bit floating-point number
`ViInt16[]`	An array of `ViInt16` values
`ViInt32[]`	An array of `ViInt32` values
`ViReal64[]`	An array of `ViReal64` values
`ViChar[]`	A string buffer
`ViConstString`	A read-only string
`ViRsrc`	An instrument driver resource descriptor (`string`)
`ViSession`	An instrument driver session handle
`ViStatus`	An instrument driver return status type
`ViBoolean`	Boolean value
`ViBoolean[]`	An array of `ViBoolean` values

To use these special user-defined data types in an instrument driver, do the following:

1. Add the VISA data types to the function panel file by using the **Options»Data Type** command in the Function Panel Editor. Then click the **Add VISA Types** button in the Edit Data Type List dialog box.
2. Include the file `vpptype.h` in the instrument driver header file.

Input and Output Parameters

Because most instrument drivers are designed to control a physical instrument, the input and output function parameters often correspond to one or more of the controls on the face of the instrument.

Define output parameters as follows:

1. Review the purpose of the function to determine the inputs and outputs.
2. Choose the data type of each parameter. The data type should be one that the application program can use easily.
   - If a parameter is an array data type, select a data type with square brackets `[]` at the end of the data type name.
   - For output parameters, select the data type of the value that users pass by reference, not the pointer to that data type. When users operate function panels interactively, LabWindows/CVI knows to pass a variable by reference because the control is defined as an output.

     For example, if a function by the name of `examp_func` has an `examp_out` integer output parameter, you prototype the function in the instrument driver include file as

     ```
     examp_func (int *examp_out);
     ```

     When you create a function panel for this function, create an output control for `examp_out` and specify its data type as `int`, not as `int *`. When a user declares variables interactively from the function panel, LabWindows/CVI creates an `int` variable and automatically puts an "`&`" in front of the variable name to pass it by reference.
3. Assign a meaningful name to each parameter.

Return Values

Instrument driver functions can also have a return value. Instrument drivers supplied by National Instruments use function return values to implement an error-handling mechanism. All instrument driver functions have a return value of type `ViStatus` (32-bit unsigned integer) that returns error and status information about the function call.

Required Instrument Driver Functions

All instrument drivers must contain functions that perform the following operations:

- *Prefix*`_init`
- *Prefix*`_close`
- *Prefix*`_error_message`

The VXI*plug&play* standard requires the following additional functions for instrument drivers that control GPIB, VXI, or serial instruments:

- *Prefix*_reset
- *Prefix*_self_test
- *Prefix*_revision_query
- *Prefix*_error_query

IVI instrument drivers must have the following additional functions:

- *Prefix*_InitWithOptions
- *Prefix*_GetErrorInfo
- *Prefix*_ClearErrorInfo
- *Prefix*_LockSession
- *Prefix*_UnlockSession
- *Prefix*_ReadInstrData
- *Prefix*_WriteInstrData

IVI instrument drivers from National Instruments also must export functions by the name of *Prefix*_IviInit and *Prefix*_IviClose, but the driver must *not* have function panels for these functions. The *Prefix*_init function calls the *Prefix*_InitWithOptions function, which in turn calls the *Prefix*_IviInit function. The *Prefix*_close function calls the *Prefix*_IviClose function.

For a description of the implementation guidelines for the required instrument driver operations, refer to the *Required Instrument Driver Functions* section.

Building the Function Tree

When users access an instrument driver from the **Instrument** menu, they can select instrument functions from one or more dialog boxes. The function tree shows the organization of the functions in dialog boxes. You use the Function Tree Editor to create the function tree.

In addition to specifying the appearance, the function tree also contains help information that the user can access from the dialog boxes. Add this help information as you create the tree.

Building the Function Panels

Users operate the function panels to execute instrument driver functions and to generate code for an application program. Each primary function requires a function panel. A secondary function can appear on one or more function panels. A function panel also can consist entirely of secondary functions. The Function Panel Editor lets you build function panels by placing controls on a blank panel in the position and order that you want them to appear.

The Function Panel Editor also lets you add online help information for each control on a panel. Add this help information as you create each panel.

Writing the Function Code

After you name the function and define its parameter list, you write the code to implement the function. LabWindows/CVI Environment describes the development tools available in LabWindows/CVI for testing and debugging your code. The instrument driver you create uses full C language source code.

To develop the instrument driver source code, follow the guidelines in Chapter 3, *Programming Guidelines for Instrument Drivers*, in the *Measurement Studio LabWindows/CVI Instrument Driver Developers Guide*.

Operating the Driver

After you create the `.c` (`.obj` or `.dll`), `.h`, and `.fp` files, you can operate the instrument driver. Load the driver using the **Instruments»Load** command and operate every function panel that you have created. Then, use the panels to generate a sample program to verify operation of the driver. Using Instrument Drivers gives more details about operating instrument drivers.

Testing the Instrument Driver

Before you distribute an instrument driver, you should fully test it. Test it from within the LabWindows/CVI interactive program and as a stand-alone application. A suggested testing sequence for instrument drivers is outlined here.

Caution Be sure to *save* copies of the original instrument source files in a separate directory.

1. Load the instrument driver and execute all functions from the function panels.
2. Verify correct operation of all functions.
3. Create and run a sample application program that exercises all the functions in the driver within LabWindows/CVI.
4. Verify correct operation of the application program.
5. Create and run a sample application that exercises all the functions in the driver within a stand-alone application.
6. Verify correct operation of the application program.

Documenting the Driver

The final step in creating an instrument driver is to document the driver. The `.doc` file describes the purpose of the driver, the function tree, and function panels. It also contains a function reference list that explains the syntax of each function in the driver. Chapter 3, *Programming Guidelines for Instrument Drivers*, in the *Measurement Studio LabWindows/CVI Instrument Driver Developers Guide*, contains guidelines and suggestions for documenting your instrument driver.

Instrument Menu

The **Instrument** menu is a *dynamic* menu. It contains a list of the loaded instrument drivers and commands to load, unload, and edit instruments. When you load an instrument, its name appears in the list. When you unload an instrument, its name disappears from the list. When you select an instrument name in the **Instrument** menu, you can access its function panels.

Load and unload instrument drivers using the commands in the **Instrument** menu.

For more information on instrument drivers, see the *Using Instrument Drivers* section.

Instrument»Load

When you select the **Load** command, a dialog box appears. In the Instrument Load dialog box, the filename `*.fp` appears in the **File Name** text box. Always load instruments through the `.fp` filename. You cannot load an instrument driver unless a `.fp` file exists for it.

When you specify a `.fp` file to load, LabWindows/CVI also looks in the same directory for a program file with the same base filename. If it finds one, it loads the instrument driver program along with the function panels.

For VXI*plug&play* instrument drivers, the program file can be in a different directory. Refer to the *Precedence Rules for Loading the Instrument Driver Program File* section for more information on loading instrument drivers.

File Format Conversion

If the `.fp` file you are loading was created using LabWindows for DOS, a message appears indicating that LabWindows/CVI is converting the `.fp` file to the current format. You can use the dialog box that appears after the conversion to save the converted `.fp` file to disk.

Instrument»Unload

When you select the **Unload** command, a dialog box appears that contains a scrollable list of all the instruments you loaded with the **Load** menu. From this dialog box, you can select one or more instrument drivers to unload.

Instrument»Edit

The **Edit** command lets you invoke the Function Panel Editor or modify the relationship between the function panel file and its associated program file. When you select **Instrument»Edit**, the Edit Instrument Dialog Box appears.

The Edit Instrument dialog box presents the following options:

- **Show Info**—Displays the names of the current function panel file and the attached program file. It also shows whether these files are in the current project and if the program file is compiled. The attached program file contains the functions that are called when users operate the function panel.
- **Attach and Edit Source**—Searches the directory that contains the function panel file for a filename that has the same prefix as the function panel file and a `.c` extension. If the file is found, a new Source window opens with the file displayed in it, and the source file is attached to the function panel. If the file is not found, you are prompted to create a new source file and a blank Source window appears.
- **Detach Program**—Detaches the program file from the function panel.
- **Reattach Program**—Attaches a program file to a function panel. It searches the directory that contains the function panel file for a filename that has the same prefix as the function panel file and a `.obj`, `.dll`, or `.c` extension. If a file is found, the program attaches it to the function panel.
- **Edit Function Tree**—Invokes the Function Tree Editor.
- **Done**—Exits the Edit Instrument dialog box without modifying the function panel.

Accessing Function Panels from the Instrument Menu

When you select an instrument name in the **Instrument** menu, the Select Function Panel dialog box appears.

The Select Function Panel dialog box shows the function panels available in the driver you selected. Class names appear in the dialog box followed by ellipses (...). The ellipses indicate that more functions or classes of functions exist below that class name. If you select **Flatten**, the list box shows all function panels at or below the current level.

If you select the **Function Names** option, the list box shows the function names associated with each function panel. While in this mode, the **Alphabetize** option redisplays the function list in alphabetical order.

If **New Window** is selected, the next selected function panel appears in a new window. Otherwise, LabWindows/CVI overwrites the current Function Panel window. This overrides the **Use Only One Function Panel Window** option in the **Environment** command of the **Options** menu.

Use **Select** to select a class name to view the functions within a class. A class can contain other classes and functions. An instrument driver can contain up to four levels of classes and functions. Each time you select a class name, the function list updates. Click on the **Up** button to return to the previous level.

When you select a function from a dialog box, that function panel appears. Refer to Chapter 6, *Using Function Panels*, for more information about function panels.

To close the dialog box without opening a function panel, select **Cancel**.

The Select Function Panel dialog box contains a **Help** button. Click on the button to get help information about the functions and classes listed in the dialog box. Select **Help** or press <F1> to display the Help dialog box for a selected function panel.

To close the Help dialog box, click on the **Done** button, press the <Enter> key, or press the <Esc> key.

Library Menu

This section explains how to use the commands in the **Library** menu. Refer to each library overview in the *Library Reference* section of the **LabWindows/CVI Help**. Use the **Library** menu commands to access function panels for the LabWindows/CVI libraries. Use library function panels to interactively run library functions and insert these function calls into any open Source window.

When you select a library name in the **Library** menu, you can access the library function panels. For more information, refer to the *Accessing Function Panels* section of Chapter 6, *Using Function Panels*.

Tools Menu

This section describes how to use the commands in the **Tools** menu.

Tools»Create ActiveX Controller

Use the **Create ActiveX Controller** command to generate a new instrument driver for an ActiveX server. When you select the **Create ActiveX Controller** command, LabWindows/CVI displays the ActiveX Controller Wizard. Refer to the *Create ActiveX Controller* topic in the **LabWindows/CVI Help** for more information.

Tools»Create ActiveX Server

You can use the LabWindows/CVI Create ActiveX Server Wizard to provide settings for your ActiveX Server project. Refer to the *Create ActiveX Server Wizard* topic in the **LabWindows/CVI Help** for more information. Also refer to the *Building ActiveX Servers in LabWindows/CVI* white paper from **Start»Program Files»National Instruments»Measurement Studio»Help»Measurement Studio Library**.

Tools»Edit ActiveX Server

Use the **Edit ActiveX Server** dialog box to create and modify objects and interfaces in your ActiveX server. Refer tothe *Edit ActiveX Server* in the **LabWindows/CVI Help** for more information. Also refer to *Building ActiveX Servers in LabWindows/CVI* from **Start»Program Files»National Instruments»Measurement Studio»Help»Measurement Studio Library**.

Tools»Create IVI Instrument Driver

Select **Tools»Create IVI Instrument Driver** to open the Instrument Driver Development Wizard, to create the source file, include file, and function panel file for controlling an instrument. You can base the new instrument driver on one of the following:

- An existing driver for a similar instrument
- The core IVI driver template
- An IVI instrument class template

The Instrument Driver Development Wizard copies the template or existing driver files and replaces all instances of the original instrument prefix with the prefix you select for your new driver.

Refer to the *Measurement Studio LabWindows/CVI Instrument Driver Developers Guide* for more information on the Instrument Driver Development Wizard.

Tools»Source Code Control

The **Source Code Control** submenu contains menu items that you can use to perform operations with your source code control system. LabWindows/CVI does not provide a source code control system. If you have a source code control system that implements the Microsoft standard Source Code Control Interface, you can attach a LabWindows/CVI project to your source code system using the Source Code Control Options dialog box. Use the Source Code Control Options command in the Project window **Options** menu to bring up the Source Code Control Options dialog box.

The exact behavior of the commands in the **Source Code Control** submenu depends on the implementation of the Source Code Control Interface that your source code control system provides and on the settings in the **Source Code Control Options** dialog box. Refer to the *Options»Source Code Control Options* section.

Some of the source code control commands bring up a Select Files dialog box when you select the commands from the Project window. Use the File Select dialog box to select the files you want to include in the operation and set any available options for the operation. Click on the **Advanced** button to set any options that your Source Code Control Interface provides for the operation.

Source Code Control commands and options appear dim if the Source Code Control Interface that source code control system provides does not support the command or option. Many of the commands transfer files between your hard disk and the Source Code Control program attached to the current LabWindows/CVI program. Depending on the command you select and the window that is active, the Source Code Control commands apply to either the list of files you select, the currently selected file in the Project window, or the file that is in the active window.

- **Get Latest Version**—Use this command to get the latest version of files from the source code control project that is attached to the current LabWindows/CVI project.
- **Get Latest Versions of All**—Use this command to get the latest version of all files in the current LabWindows/CVI project.
- **Check Out**—Use this command to check out files from the source code control project attached to the current LabWindows/CVI project.
- **Check In**—Use this command to check files into the source code control project attached to the current LabWindows/CVI project.
- **Undo Check Out**—Use this command to undo check out actions you previously performed on files.
- **Add File to Source Control**—Use this command to add files to the source code control project that is attached to the current LabWindows/CVI project.
- **Remove From Source Control**—Use this command to remove files from the source code control project that is attached to the current LabWindows/CVI project.
- **Show History**—Use this command to view the source code control history for a file.
- **Show Differences**—Use this command to view the differences between a file on disk and the latest version of the file in the source code control system.
- **Properties**—Use this command to view the source code control properties and status of a file.
- **Refresh Status**—Use this command to have LabWindows/CVI update the source code control status of files. If you use the source code control system GUI to change the status

of files in your source code control system, LabWindows/CVI might not know about these changes until you select **Refresh Status**.

- **Source Code Control**—Use this command to launch the GUI interface provided by your source code control system.
- **Clear Source Code Control Error Window**—Use this command to clear the contents of the Source Code Control Error window.

Tools»Source Code Browser

The Source Code Browser is a cross-reference tool that lists all files, functions, variables, data types, and macros in your program. You can use the browser to identify how different parts of a program interact with each other. The **Identifier/Filename** box lists the last eight items accessed. Browse information is not available in Release configuration.

To access the Source Code Browser, select **Tools»Source Code Browser** or press <Ctrl-F1> while placing the cursor over functions, variables, data types, and macros in your code. You also can enable the **Generate source code browse information** option in the Build Options dialog box. **Generate source code browse information** generates browse information in the Source Code Browser window at compile time.

You can find a definition by selecting **Edit»Go to Definition**, go to the next reference by selecting **Edit»Go to Next Reference**, or return to the previous location by selecting **Edit»Go Back**. You also can type a substring or the entire name of the file, function, variable, data type, or macro in the **Identifier/Filename** box. Wildcards are not supported, and the search is not case sensitive. All search matches appear in the **Matches Found** box.

Browsing on Files

Under the Views column, you can choose one of the following:

- Functions—the functions defined in the file. The line number where the function exists in the code appears next to the function name. Static functions are identified.
- Included files—a hierarchical display of included files in the file.
- Macros—the macros in the file. The line number where the macro exists appears next to the macro name.
- Referencing—a hierarchical list of variables and functions, grouped according to module, explicitly referenced in the file.
- Referenced from—the functions and variables, grouped according to module, referenced from another file.
- Types—the types defined in the file.
- Variables—the variables defined in the file. Line numbers appear next to the variable name. Static variables also are identified next to the line number.

Browsing on Functions

Under the Views column, you can choose from the following:

- Definition—the file that contains the definition for the function, along with the line number in the code.
- References—a hierarchical list of files where the function is referenced.
- Calling—list of functions that the function is calling and the number of references.
- Calling from—a hierarchical list of functions that call this function.

Browsing on Variables, Data Types, and Macros

Under the Views column, you can choose from the following:

- Definition—a list of file locations where the variable, data type, or macro is defined.
- References—the files and line numbers where you can find references to the variable, data type, or macro.

Tools»UI to Code Converter

Use this utility to generate code that programmatically creates the panels and menu bars in the user interface resource (`.uir`) files you select. Refer to the help dialog box in the UI to Code Converter utility for more information.

The UI to Code Converter is a LabWindows/CVI application that reads the contents of `*.tui` files (alternate format `*.uir` files) and generates code to build a UI programmatically, eliminating the need to distribute a separate file with your LabWindows/CVI application. The UI to Code Converter works best on `*.tui` files created with LabWindows/CVI v. 5.5.

Using the UI to Code Converter Utility

You specify all code-generation parameters from the UI to Code Converter's main panel. The first step is to select panels and menubars for which you would like to generate code. With LabWindows/CVI 4.0 or greater, you may use the UI Editor to save UI Resources in text format, as `*.tui` files. Select **Options»Save In Text Format** to save a UIR file as a TUI file. When you select "Add" from the main panel, you will be prompted to specify a `*.tui` file. Once you select a file, a dialog will appear listing all of the UI objects (panels or menubars) saved in that `*.tui` file. Check any objects for which you would like to generate code, and click "OK". Any panels and menubars you select in this way will appear in the two list boxes on the main panel.

Each UI panel or menubar you select will be given a default function name, which you may change. UI to Code Converter will generate these functions (one per UI Object) for you. You may call these functions in your application in place of `LoadPanel` or `LoadMenubar`, and the associated UI object will be created programmatically. The functions return an object

handle or standard UI error code, and take a parent object handle as an input. The use of the functions is identical to that of `LoadPanel` and `LoadMenubar`, with the exception that you do not specify a `*.uir` file or constant name.

The main panel allows you to specify Output Targets, either files, the STDIO Window, or both. When you select "Build Files," you must specify a `*.h` and a `*.c` output file. UI to Code Converter will output all generated functions to the specified source file, and place necessary includes and definitions in the header file.

You can also specify some optional parameters for code-generation. UI to Code Converter is intelligent enough to skip code-generation for setting attributes that are default values. For example, if your UIR has a visible command button, it is not required to specifically set the VISIBLE attribute when building that control programmatically, it is the default value. You may specify whether to skip default attribute calls altogether, or you may choose to have the generated code show default attributes as commented code. You may also specify what type of error-checking you would like your functions to use—the standard `if-then` construct or the Programmer's Toolbox "errChk" macro. With all of the options, you will be able to see the result by quickly glancing through the generated code.

Using the Generated Code

UI to Code Converter will generate one function per UI object, and that function may be used in place of `LoadPanel` or `LoadMenubar` to generate the object rather than load it from an external file. The output source file you specified will contain these function definitions. The output header file will prototype all of the functions and declare identifiers corresponding to what were previously the control constant IDs. In other words, the constant names defined in the header file generated automatically when you saved a `*.uir` file will now be integer identifiers. This means that the rest of your application code stays the same when using the generated code in place of `LoadPanel` and `LoadMenubar`. To gain access to the identifiers and avoid conflicts with the constant definitions, simply replace the header file associated with the UIR file with the generated header file.

The following outlines the general procedure for using generated UI code in an application that originally used a standard `*.uir` file:

1. From the UIR editor, save any `*.uir` files used by your application as `*.tui` files.
2. Run UI to Code Converter, select the appropriate objects from the `*.tui` files, and verify or adjust the function names.
3. Select an output header and source file, and specify the code-generation options.
4. Generate all code by selecting **Generate Code**.
5. In your application's modules, #include the generated header file in place of the original header file associated with the UIR (`UIRNAME.H`)
6. Add the output source file to your project.

7. Replace all calls to `LoadPanel` and `LoadMenubar` with calls to the generated functions defined in the output source file. These functions return the object's handle or an error code just like `LoadPanel` and `LoadMenubar`, and also take as inputs the handles of any parent panel (or 0 if the panel is to be top-level).
8. Run your application, it will behave as before, but you are no longer dependent on an external file.

In general, you will develop your application using the UIR Editor and loading UI objects from files as you have done before. When you are ready to distribute the application, go through the above steps and eliminate the external dependency. If you want to modify the UI after generating the code, you may either edit the generated code or use the UIR Editor, re-save the `*.tui`, and re-generate the function definitions.

UI to Code Converter Limitation

The UI to Code Converter application will handle any UIR transparently, with one restriction. The generated code will declare identifiers for what were originally #defined constants. Since you cannot switch on identifiers, you will have to replace any switch-case structures that relied on the constant definitions with a series of if constructs. For example, you might have been switching inside a callback function on the "control" parameter. You would need to replace this with a series of `if` constructs.

Tools»User Interface Localizer

Use this utility to translate your user interface resource files (`.uir`) into other languages. Refer to the help dialog box in the User Interface Localizer utility for more information.

This utility allows you to create a language resource file (`*.lwl`), which you can use with `LoadLocalizedPanel` to translate a `.uir` file. When you load a `.uir` or `.tui` file, you will see a tree populated with all the strings of the UI objects in the file. For each string, the utility maintains a default and a local string. The strings stored in the UIR or TUI file are said to be in the default language. When you highlight one of the string items in the tree, you will see the default language string, if there is one, in the "Default Language" textbox. Enter the local language string in the "Local Language" textbox. When you have finished translating all the strings, choose "Export Language" in the file menu to create the language resource file.

In the tree, strings without translations are marked by bold text. A string is considered untranslated if there is a default language string but no local language string. An item that does not have a default language string may have a local language string, but it is never considered untranslated. If an item does not have a local language string, the default language string is used.

You can preview the active panel or menubar to see the translated strings on the panel.

If addition to saving language resource files, you can save and import dictionary files (`*.lwd`). A dictionary file is simply a list of local and default language string pairs. The dictionary file has no affiliation with any particular file, so you can apply it to any UIR or TUI file.

Once you have a language resource file, you can use the functions `LoadLocalizedPanel` and `LoadLocalizedMenubar` in `toolslib\localui\localui.fp`.

Tools»Convert UI to Lab Style

Use the **Convert UI to Lab Style** command to convert Classic Style controls to Lab Style controls. Converting to Lab Style controls does not change control settings you have specified.

When you select this command, the UIR Conversion dialog box appears.

- **Original .UIR**—Use the **Load** button to browse to the `.uir` you want to convert.
- **New .UIR**—Enter the path and name of the new `.uir`. By default, LabWindows/CVI uses the same path and name of the original `.uir`.
- **Backup Original .UIR**—Enable to create a backup of the original `.uir`. LabWindows/CVI creates the backup in the original location.
- **Convert UIR**—Converts the `.uir`.
- **Quit**—Cancels the operation and closes the UIR Conversion dialog box.

User Defined Entries in the Tools Menu

You can install your own entries in the **Tools** menu. Each entry invokes an executable with optional command-line arguments. Select **Options»Tools Menu Options** in the Project window to add your own entries to the **Tools** menu.

Window Menu

Use commands in the **Window** menu to bring any open window to the front for viewing or editing.

Window»Cascade Windows

Use this command to arrange all open windows so that each title bar is visible.

Window»Tile Windows

Use this command to arrange all open windows in smaller sizes to fit next to each other.

Window»Minimize All

The **Minimize All** command hides all the LabWindows/CVI windows, including the Project window. You can restore the windows by clicking on LabWindows/CVI in the Windows task bar.

Window»Close All

The **Close All** command closes all the LabWindows/CVI windows, excluding the Project window.

Window»Project

Use this command to bring the Project window to the front.

Window»Build Errors

If you attempt to build a project and the project has build errors, such as syntax or link errors, the Build Errors window contains a list of the errors. To bring the Build Errors window to the front for viewing, select **Window»Build Errors**.

Window»Run-Time Errors

If you attempt to run a project and the project has run-time errors, such as over-indexing an array, the Run-Time Errors window contains a list of the errors. The Run-Time Errors window also displays the output of the Utility Library `ErrorPrintf` function. To bring the Run-Time Errors window to the front for viewing, select **Window»Run-Time Errors**.

You can use the Utility library `DebugPrintf` function to send debug output to the Debug Output window.

Window»Debug Output

LabWindows/CVI displays the output of the Utility library `DebugPrintf` function and the Windows SDK `OutputDebugString` function in the Debug Output window.

It is a good idea to use the `DebugPrintf` function for all your debug output strings. Unlike the Standard I/O window, you can access and scroll through the Debug Output window even while your program is suspended. Enable the **Bring Debug Output Window to Front Whenever Modified** option in the Environment Options dialog box to make LabWindows/CVI bring the Debug Output window to the front whenever your program adds text to the window.

Window»Source Code Control Errors

This window displays warnings and errors that source code control systems return when you execute commands from the **Source Code Control** submenu of the **Tools** menu.

Window»Memory Display

The **Memory Display** command opens the Memory Display window that you can use to view and edit the memory of the program you are debugging. This window allows you to view and edit the data in hexadecimal (byte, word, long), decimal (byte, word, long), single-precision floating point, double-precision floating point, or ASCII representation. You must enable the Edit Mode option before you can modify the process' memory. To change the value of a memory location, click on that cell in the Memory Display window and type in the new value.

Window»Variables

Use this command to bring the Variables window to the front. The Variables window shows the contents of all variables currently defined in LabWindows/CVI. You can access the Array Display and String Display windows from the Variables window.

The Variables window is useful for debugging programs. LabWindows/CVI updates the Variables window at each breakpoint, and you can modify the variables while in a breakpoint state.

For more information about the Variables window, see the *Variables Windows* section of Chapter 10, *Variables and Watch Windows*.

Window»Watch

The **Watch** command brings the Watch window to the front. The Watch window shows a set of variables and expressions that you specify. You can access the Array Display and String Display windows from the Watch window.

The Watch window is useful for debugging programs. Watch variables and expressions update at each breakpoint unless you set them to update continuously.

For more information about the Watch window, refer to the *Watch Window* section of Chapter 10, *Variables and Watch Windows*.

Window»Interactive Execution

Use the **Interactive Execution** command to bring the Interactive Execution window to the front. Unlike the Source window, you can execute incomplete programs in the Interactive Execution window. For example, you can execute variable declarations and assignment statements in C without declaring a `main` function. Refer to the *Interactive Execution Window* section of Chapter 5, *Source and Interactive Execution Windows*, for more information.

Window»Open Source Files

The **Window** menu lists open source files at the bottom. If the file is in the project, only the filename appears. If the file is not in the project, the full pathname appears. Select a source file from this menu to bring its window to the front. For more information about the Source window, refer to the *Source Windows* section of Chapter 5, *Source and Interactive Execution Windows*.

Options Menu

You use commands in the **Options** menu to set up preferences in the LabWindows/CVI environment, and execute various utilities.

Options»Build Options

You can set the LabWindows/CVI compiler options by selecting **Options»Build Options** in the Project window. This command opens a dialog box that allows you to set the following LabWindows/CVI compiler options:

- **Compatibility With**—This option displays the current compiler compatibility mode. For more information on external compiler compatibility, refer to Chapter 3, *Compiler/Linker Issues*, in the *Measurement Studio LabWindows/CVI Programmer Reference Manual*.
- **Default Calling Convention**—This option sets the compiler's default calling convention. For both compilers, the default calling convention is normally `__cdecl` but can be changed to `__stdcall`. Do not change the default calling convention to `__stdcall` if you plan to generate static library or object files for both compatible external compilers. Refer to Chapter 3, *Compiler/Linker Issues,* in the *Measurement Studio LabWindows/CVI Programmer Reference Manual* for more information.

Note If you want to create an object file, static library file, or DLL that exports functions with the `__stdcall` calling convention, it is a good idea to explicitly declare the functions as `__stdcall` in the include (`.h`) file and the source (`.c`) file rather than relying on the **Default Calling Convention** option. If you do not explicitly declare the functions as `__stdcall` in the include file and if another developer uses the object file, library file, or DLL in LabWindows/CVI or an external compiler without setting the default calling convention to `__stdcall`, the functions do not work correctly.

- **Maximum Stack Size (bytes)**—Your program uses the stack for passing function parameters and storing automatic local variables. By setting a stack limit, LabWindows/CVI can catch infinitely recursive functions and report a stack overflow. Refer to Chapter 1, *LabWindows/CVI Compiler*, of the *Measurement Studio LabWindows/CVI Programmer Reference Manual* for limitations on the stack size.

If you enable the **Detect uninitialized local variables at runtime** option, LabWindows/CVI might use extra stack space. You can adjust your maximum stack size to avoid a stack overflow.

- **Image Base Address**—This option specifies the address where LabWindows/CVI loads a DLL or executable into memory. By specifying the image base address, you can avoid relocating DLLs, which can slow down the load time of DLLs. You also can avoid collisions, which occur when LabWindows/CVI attempts to load more than one DLL with the same address. You can specify any value, or you can select the following default values:
  - x00400000 for executables
  - x10000000 for DLLs
- **Debugging Level**—LabWindows/CVI uses this setting only if you check the **Debug** item in the **Configuration** submenu of the Project window **Build** menu. If you check the **Release** item in the **Configuration** submenu, LabWindows/CVI compiles all source files (`*.c`) without debugging information. Refer to the *Build»Configuration* section for information on the Configuration submenu.

  The following three debugging levels exist for the source modules in your application.
  - **No Run-time Checking**—In this mode, you can set breakpoints and use the Variables window. You have no protection from run-time memory errors, and you cannot use the **Break on Library Errors** option.
  - **Standard**—In this mode, you can set breakpoints, use the Variables window, and use the **Break on Library Errors** option. You also have protection from run-time memory errors. Refer to Chapter 1, *LabWindows/CVI Compiler*, of the *Measurement Studio LabWindows/CVI Programmer Reference Manual* for more information.
  - **Extended**—This mode has the same benefits as Standard mode, with added user protection that validates every attempt to free dynamically allocated memory by verifying that the address you pass is actually the beginning of an allocated block.
- **Detect Uninitialized Local Variables at Run-Time**—This option checks for the run-time use of variables that do not have values assigned to them. Enabling this option causes LabWindows/CVI to use extra stack space. To avoid a stack overflow, adjust the maximum stack size.
- **Track Include File Dependencies**—This option keeps the project up-to-date by tracking the dependencies between source files and include files. Whenever you modify a file, LabWindows/CVI marks for compilation all source files that include the modified file.
- **Prompt for Include File Paths**—This option sets LabWindows/CVI to prompt you to make a manual search for any header files listed in the `#include` lines that the compiler cannot find. When you find them, you can automatically insert the appropriate path into the Include Paths list for the project.

- **Display Status Dialog Box During Build**—This option displays a status dialog box during the build that shows the name of the file being compiled, the number of errors and warnings encountered, and a percent completed value. Your project compiles faster when you disable this feature.
- **Make 'O' Option Compatible with CVI 5.0.1**—When you enable this option, the 'O' option in the Project Window performs as it did in CVI 5.0.1. LabWindows/CVI compiles the file without debugging information and writes the `.obj` file to disk in the same directory as the source file.
- **Uninitialized Local Variables Detection**—This option generates error messages when you access variables that do not have values assigned to them. You can select the following levels:
  - **Disabled**—This mode disables uninitialized local variable warnings and errors.
  - **Conservative**—This mode flags only variables that do not have values assigned to them.
  - **Aggressive**—This mode flags all variables that may or may not have values assigned to them.
- **Detect Signed/Unsigned Pointer Mismatches**—This option generates a compiler warning for pointer assignments in which the left side and right side are not both `signed` or `unsigned` expressions. According to the ANSI C standard, these assignments are errors because they involve incompatible types. In practice, however, such assignments cause no problems.

  The LabWindows/CVI compiler checks assignment statements and function call arguments to ensure that the `lvalue` and `rvalue` expressions have compatible types. If you enable the **Enable Signed/Unsigned Pointer Mismatch Warning** option, LabWindows/CVI generates compile warnings when the `lvalue` and `rvalue` expressions are both pointers to integers but one points to a signed integer and the other points to an unsigned integer. For example, the LabWindows/CVI compiler generates a `signed type mismatch between pointer to char and pointer to unsigned char` warning on the call to `MyFunction` in the following code example.

  ```
  void MyFunction (unsigned char *x);

  char *y = "my string";

  main () {

      MyFunction (y);

  }
  ```

- **Detect Unreachable Code**—This option generates a compiler warning for statements that the compiler cannot reach on execution. When you enable the **Detect Unreachable Code** option, the LabWindows/CVI compiler generates a warning at each line of code

that cannot be reached during the execution of your program. For example, LabWindows/CVI reports a warning on the break statement in the following code.

```
switch (intval) {
    case 4:
    return 0;
    break;
}
```

- **Detect Unreferenced Identifiers**—This option generates compiler warnings for identifiers that are not referenced in your program.
- **Detect Assignments in Conditional Expressions**—LabWindows/CVI checks for possible errors in conditional expressions that can make the conditional expression an assignment.
- **Require Function Prototypes**—This option requires you to precede all function references with a full prototype declaration. A full prototype includes the function return type and the types of each parameter. If a function has no parameters, a full prototype must have the void keyword to indicate this case. A new style function definition (one in which you declare parameters directly in the parameter list) can serve as a prototype.

  Missing prototype errors can occur at the following places:

  – Typedefs such as `typedef void FUNTYPE()`
  – Function pointer declarations such as `void (*fp)()` whether used as a global, local, parameter, array element, or structure member
  – Old style function definitions, in which you declare parameters outside of the parameter list, that you do not precede with a full prototype
  – Function call expressions such as `(*fp)()`, where fp does not have a full prototype

Caution It is best to *enable* the Require Function Prototypes option. If disabled, some of the run-time error checking is also disabled.

- **Require Return Values for Non-void Functions**—This option generates compile warnings for non-void functions, except `main`, that do not end with a `return` statement that returns a value. LabWindows/CVI reports a run-time error when a non-void function executes without returning a value.

For example, the following code always produces a compile-time warning, and it produces a run-time error when `flag` is FALSE.

```
int fun (void) {
    if (flag) {
        return 0;
    }
}
```

- **Maximum Number of Compile Errors**—This option sets an upper limit on the number of compile errors LabWindows/CVI lists in the Build Errors window for each source file.
- **Stop on First File with Errors**—This option sets the LabWindows/CVI compiler to terminate compilation after it finds one file with errors. Using this option, you can correct build errors in your project one file at a time.
- **Show Build Error Window for Warnings**—This option sets the LabWindows/CVI compiler to open the Build Error window when warnings occur, even if no errors exist. If you deactivate it, warnings can occur without being brought to your attention.
- **Generate Source Code Browse Information**—Use this option to view source code browse information under **Tools»Source Code Browser**.

Options»Compiler Defines

The LabWindows/CVI compiler accepts compiler defines through the **Compiler Defines** command in the **Options** menu of the Project window.

Compiler defines have the following syntax:

`/Dx or /Dx=y`

The variable `x` is a valid C identifier. You can use *x* in your source code as a predefined macro. For example, you can use *x* as the argument to the `#if` or `#ifdef` preprocessor directive for conditional complication. If `y` contains embedded blanks, you must surround it with double quotation marks.

The Compiler Defines dialog box contains a list of the macros that LabWindows/CVI predefines. This list includes the name and value of each predefined macro. LabWindows/CVI predefines the macros to help you write platform-dependent code.

Note The default Compiler Defines string contains the following definition:
`/DWIN32_MEAN_AND_LEAN`

This definition reduces the time and memory taken when compiling Windows SDK include files. For more information, refer to Chapter 3, *Compiler/Linker Issues,* in the *Measurement Studio LabWindows/CVI Programmer Reference Manual*.

Predefined Macros

CVI is defined to be 1 in LabWindows/CVI version 3.0, 301 in version 3.0.1, and 310 in version 3.1, and so on.

- _NI_mswin_.
- _NI_mswin32_.
- _NI_i386_.
- _CVI_DEBUG_ is defined if the **Debug** item is checked in the Configuration submenu of the Project window **Build** menu. The value of the macro is 1.
- _CVI_EXE_ is defined if the project Target Type is **Executable**.
- _CVI_DLL_ is defined if Target Type is **Dynamic Link Library**.
- _CVI_LIB_ is defined if Target Type is **Static Library**.
- _LINK_CVIRTE_ is defined if the project's Runtime Support is **Full Runtime Engine**. The target will link to cvirte.dll.
- _LINK_INSTRSUP_ is defined if the project's Runtime Support is **Instrument Driver Only**. The target will link to instrsup.dll.
- _LINK_CVI_LVRT_ is defined if the project's Runtime Support is **LabVIEW Real-Time Only**. The target will link to cvi_lvrt.dll.
- __DEFALIGN is defined to the default structure alignment: 8 for Microsoft; 1 for Borland.
- _NI_VC_ is defined to 220 if in Microsoft Visual C/C++ compatibility mode.
- _NI_BC_ is defined to 451 if in Borland C/C++ compatibility mode.
- _WINDOWS.
- WIN32.
- _WIN32.
- __WIN32__.
- __NT__.
- _M_IX86 is defined to 400.
- __FLAT__ is defined to 1.

Note LabWindows/CVI does not define _MSC_VER or __BORLANDC__. The external compilers each define one of these macros. If you port code originally developed under one of these external compilers to LabWindows/CVI, you might have to manually define one of these macros.

Options»Include Paths

The **Include Paths** command invokes a dialog box in which you can list paths that the compiler uses when searching for header files with simple pathnames. The Include Paths dialog box has two lists of paths in which to search for include files. LabWindows/CVI saves the top list with the project file. LabWindows/CVI saves the bottom list from one session to another on the same machine, regardless of the project. Refer to Chapter 1, *LabWindows/CVI Compiler*, of the *Measurement Studio LabWindows/CVI Programmer Reference Manual* for a description of the entire include path search precedence.

Options»Instrument Directories

Use the **Instrument Directories** command to list directories that LabWindows/CVI can search for instrument drivers on which other instrument drivers depend. The `.fp` files of the dependent drivers store the names of the `.fp` files of the drivers on which they depend. When loading an instrument `.fp` file that references other instrument `.fp` files, LabWindows/CVI tries to find the referenced instrument `.fp` files and load them. It searches for each `.fp` file in the following directories and in the following order:

1. The directory of the referencing `.fp` file
2. The directories listed in the Instrument Directories dialog box
3. The subdirectories under the `cvi\toolslib` directory
4. The `cvi\instr` directory

LabWindows/CVI saves the Instrument Directories list from one session to another.

To find out more about referencing one instrument from another, refer to Chapter 7, *Function Tree Editor*, for more information.

You also use the Instrument Directories list when you load a project file that you have moved since you last saved it. If a `.fp` file listed in the project cannot be found using either its original pathname in the project or its location relative to the project, LabWindows/CVI searches the Instrument Directories list for a `.fp` file with the same base name.

Options»Run Options

The **Run Options** command invokes a dialog box you use to set the following options:

- **Save Changes Before Running**—You can configure LabWindows/CVI to never save modified files, to always save modified files, or to ask whether to save modified files before running.
- **Break on Library Errors**—When you enable this checkbox, a breakpoint occurs when any function call to a LabWindows/CVI library results in an error.
- **Break On First Chance Exceptions**—This option determines what LabWindows/CVI does when your program causes an exception. If you enable this option,

LabWindows/CVI suspends your program before it can catch and handle the exception. In general, disable this option if you are debugging code that generates exceptions that your code catches and handles. If LabWindows/CVI suspends your program before your program catches the exception, you can use the Continue command to resume execution of your program.

Note When you use LabWindows/CVI to debug a DLL that you are calling from National Instruments TestStand, enable this option so that LabWindows/CVI can suspend your program before TestStand catches the exception.

- **Hide Windows**—When you enable this checkbox, the LabWindows/CVI environment hides all its windows until execution terminates or a breakpoint occurs.
- **Enable Global Ctrl+F12 Debug Break Key**—Use this option to install a system-wide hot key to suspend the execution of your LabWindows/CVI program or the interactive window. You can use this hot key when LabWindows/CVI or your program is not the active application. Enabling this option might interfere with other applications (including your program) that use <Ctrl+F12> for different purposes. It also might not work correctly when you are debugging two applications with LabWindows/CVI at the same time. The hot key is active only when LabWindows/CVI is running a user program or the interactive window.

Options»Command Line

Use the **Command Line** command to enter the command-line arguments for your program. When you run your program in the LabWindows/CVI environment, LabWindows/CVI passes the command-line arguments to your `main` function in the **argc** and **argv** parameters.

If your project makes a DLL, you can pass command-line arguments to an external program that you run to debug the DLL. Specify the external program pathname and command-line arguments using the **Select External Process** command in the **Run** menu. The same command-line arguments appear when you select the **Command Line** command from the **Options** menu.

Options»Environment

The **Environment** command invokes a dialog box you use to set the following options:

- **CVI Environment Sleep Policy**—Each time LabWindows/CVI checks an event from the operating system, it can put itself in the background, in sleep mode, for a specified period of time. While LabWindows/CVI is in sleep mode, other applications have more processor time. However, LabWindows/CVI might run slower. You can specify how much LabWindows/CVI sleeps. You have the following sleep policy choices.
  - Do not sleep
  - Sleep some (sleep a short period of time)

  – Sleep more (sleep a longer period of time, the default setting)

  The setting that is optimal for you depends on the operating system you are using and the other applications you are running. Generally, for Windows National Instruments recommends the sleep more mode. If you think you might have to make an adjustment, try the different settings and observe the resulting behavior.

- **Interactive Window Memory Size**—Use this control to set the amount of memory you want LabWindows/CVI to reserve for executing function panels or code in the Interactive window. If LabWindows/CVI displays the message **`"Insufficient memory for Interactive window"`**, you must increase the value in this control, select **Clear Interactive Declarations** from the **Build** menu in the Interactive Execution window, and then execute the function panel or interactive window statements again.
- **Lines in Debug Output Window**—Shows the last *n* number of lines that you specify.
- **Use Only One Function Panel Window**—Enable this option to overwrite the current function panel window each time you select a new function panel window.
- **Use Only One Browse Info Window**—Enable this option to overwrite the current browse info window each time you select a new file, function, variable, data type, or macro.
- **Go to Source After Inserting Code from Function Panel**—Enable this option if you want LabWindows/CVI to close the function panel window after you execute the **Insert Function Call** command.
- **Check Foreground Lockout Setting on Startup (Win2000/98 only)**—Windows 2000 and Windows 98, by default, sometimes prevent applications from bringing their windows to the front. This behavior prevents LabWindows/CVI from bringing its windows to the front when you debug your programs. Enable this option if you want LabWindows/CVI to prompt you at startup if the system configuration does not allow LabWindows/CVI to bring its windows to the front. The prompt allows you to change the system setting.
- **Enable Data Tool Tips**—Enable this option if you want LabWindows/CVI to display the values of variables and selected expressions when you place the mouse cursor over the variables or selected expressions in a Source window.
- **Use Console Window for Standard I/O When Debugging**—Enable this option if you want LabWindows/CVI to use the Microsoft DOS console window for displaying output and receiving input while you debug a project.
- **Bring Debug Output Window to Front Whenever Modified**—Enable this option if you want LabWindows/CVI to bring the Debug Output window to the front whenever your program adds text to the window.
- **Force Loaded Instrument Driver's Source into Interactive Window**—Enable this option if you run function panel or Interactive window code that uses `LoadExternalModule` or `LoadExternalModuleEx` to load the source file for an instrument driver that is already loaded in the **Instrument** menu.

- **Force Project Compiled Source Files into Interactive Window**—Enable this option if you run function panel or Interactive window code that uses `LoadExternalModule` or `LoadExternalModuleEx` to load a source file that is in the project.

Options»Library Options

You use the **Library Options** command to specify optional National Instrument libraries that load automatically when you start LabWindows/CVI. You also can specify which of the optional National Instruments user libraries to load on startup using the **Library Options** command. The **Library Options** command invokes the Library Options dialog box, which contains National Instruments Libraries and User Libraries.

National Instruments Libraries

There are five optional National Instruments libraries:

- Advanced Analysis
- Easy I/O for DAQ
- Data Acquisition
- VXI
- GPIB/GPIB 488.2

In the National Instruments section, click on the selection box to the left of the library name(s) that you want LabWindows/CVI to load at startup. Use the National Instruments Libraries section of the Library Options dialog box to specify whether or not to load these libraries into memory when you start LabWindows/CVI. A checkmark next to the library name indicates that you want the library to be loaded. Changes do not take effect until the next time you launch LabWindows/CVI.

If you do not load a library, you cannot call any of the functions in that library and you cannot access any of its function panels.

If LabWindows/CVI fails to load a requested library, it is probably because LabWindows/CVI cannot find the appropriate files. The files in Table Libraries in the `bin` Directory of LabWindows/CVI belong in the `bin` directory of the LabWindows/CVI installation directory.

Library	Windows
Analysis or Advanced Analysis	`analysis.lib`
Data Acquisition	`dataacq.lib`
Easy I/O for DAQ	`easyio.lib`

Library	Windows
VXI	`nivxi.lib`
GPIB/GPIB 488.2	`gpib.lib`

You must also install the drivers that came with your Data Acquisition, VXI, and GPIB/GPIB 488.2 hardware to use the optional libraries.

User Libraries

You can install your own libraries into the **Library** menu. A user library has the same form as an instrument driver. Anything that can be loaded into the **Instrument** menu can be loaded as a user library, provided the program is in compiled form. Refer to the *Using Instrument Drivers* section for more information on loading files with the **Instrument** menu.

The main difference between modules you load as instrument drivers and those you load as user libraries is that you can unload instrument drivers with the **Unload** command in the **Instrument** menu, but you cannot unload user libraries. Also, because user libraries must be in compiled form, you cannot edit them when they are in the **Library** menu. Refer to the *Measurement Studio LabWindows/CVI Instrument Driver Developers Guide* for detailed information about writing an instrument driver.

You install user libraries by selecting **Options»Library Options** in the Project window. Once a library is installed, the next time you launch LabWindows/CVI the libraries will load automatically and appear at the bottom of the **Library** menu.

Dummy .fp Files for Support Libraries

If you develop a library module to provide support functions for the modules in your project, you can install it as a user library. By doing so, you ensure that the library is always available in the LabWindows/CVI development environment. If you do not want to develop function panels for the library, create a `.fp` file without any classes or functions. In that case, LabWindows/CVI loads the library at startup but does not include the library name in the **Library** menu.

Options»Tools Menu Options

Use the **Tools Menu Options** command to add your own menu items to the **Tools** menu. Each entry consists of a menu item name and an associated command line to execute. Each command line consists of a program name and optional arguments. When you execute an item from the **Tools** menu, LabWindows/CVI calls a system function to start the program as another process.

- **Menu Item Name**—A list box that contains your current **Tools** menu entries.
- **Command Line**—An indicator that shows the command line for the currently selected menu item name.
- **Add**—Click on this button to add a new entry.
- **Edit**—Click on this button to edit the selected entry.

The **Add** and **Edit** buttons open the Add/Edit Tools Menu Item dialog box, where you can set the following options:

- **Menu Item Name**—The string that appears in the **Tools** menu. To specify an accelerator key for the menu item, insert two underscores in front of the designated letter.
- **Program Name**—The pathname of the program to execute when you select the menu entry. You can specify full pathname, simple filename, or a relative pathname. You can use the **Browse** button to look for the program on disk.
- **Command Line Arguments**—Arguments you want to pass to the program. You can leave this entry blank.

LabWindows/CVI saves your **Tools** menu entries from one session to another, not in the project.

Options»Source Code Control Options

The **Source Code Control Options** command brings up the Source Code Control Options dialog box, where you can set the following options:

- **Use Global Source Code Control settings**—Enable this option to use the default global settings.
- **Use Project Source Code Control settings**—Enable this option to use project-specific options. Individual project options override global settings.
- **Provider**—The name of the source code control system that contains the source code control project attached to the currently loaded LabWindows/CVI project.
- **Project**—The name of the source code control project attached to the currently loaded LabWindows/CVI project.
- **Attach**—Use this button to attach an existing source code control system project to the current LabWindows/CVI project. After you attach a source code control provider, this button changes to **Change**. Use **Change** to change source code control system projects.
- **Create**—Use this button to create a new source code control system project and attach it to the current LabWindows/CVI project.

- **Perform Same Actions for .h File as for .uir File**—Because LabWindows/CVI generates a `.uir` header file each time the uir file is saved, it is a good idea to perform the same source code actions on the header file as you perform on the `.uir` file.
  - **Ask**—When you perform a source code control action, LabWindows/CVI asks if you want to perform the same action on the header file and on the associated `.uir` file.
  - **Always**—When you perform a source code control action, LabWindows/CVI automatically performs the same action on the header file and on the associated `.uir` file.
  - **Never**—When you perform a source code control action, LabWindows/CVI does not perform the same action on the header file and on the associated `.uir` file.
- **Do Not Include .prj File in Source Code Control Actions**—Enable this option if you do not want LabWindows/CVI to automatically include the LabWindows/CVI project file in source code control operations. This options always applies to the **Get Latest Version of All** command. It applies to other commands when they are executed from the Project window and no project files are selected.
- **Suppress CVI Error Messages**—Some source code control systems display their own dialog boxes when errors occur during a source code control operation. Enabling this option suppresses all LabWindows/CVI error dialog boxes displayed when an error is reported by the source code control system.
- **Always Show Confirmation Dialog**—Displays a dialog box where you can confirm your source code control actions.
- **Use Default Comment**—Enable this option if you do not want to be prompted for a comment when you check in or add files to a source code control project. Enter the comment that you want LabWindows/CVI to pass to the source code control system.
- **Use Default Username**—You can change your username by enabling this option and entering a new name.
- **Advanced**—Click this button to access advanced options provided by your source code control system. If your source code control system does not provide advanced options, this control is dimmed.

Options»Font

Use the **Font** command to select the font and font size for the text in the Project window.

Options»Colors

Use the **Colors** menu item to select colors for the Project window, Source window, Interactive Execution window, Standard I/O window, Watch window, Variables window, String Display window, and Array Display window. The **Colors** menu item does not affect dialog boxes, function panels, and the User Interface Editor window.

The **Colors** menu item opens a dialog box that contains a list box. Each line in the list box describes the purpose of the color and shows its current color state. To change the color, click and hold on the color control at the bottom left of the dialog box. A color pop-up palette appears. While holding the mouse button down, move the pointer over the desired color and then release. The color change takes effect immediately in all LabWindows/CVI windows that are currently visible.

To change all the colors to their default state, click on **Default**. All currently visible LabWindows/CVI windows immediately reflect the color changes.

If you want to accept these changes, click on the **OK** button. If you want to revert to the state before the dialog box appeared, click on **Cancel**.

You also can access the **Colors** command in the Source window and the Interactive Execution window.

The Color dialog box also contains eight color types for syntax coloring. Refer to the *Options»Syntax Coloring* section in Chapter 5, *Source and Interactive Execution Windows*.

When you enable the **Use System Colors** option, several color types associated with the Project window, Source window, and scroll bars disappear from the list box. LabWindows/CVI automatically assigns colors to these types based on the system colors you set in the **Appearance** tab in the Windows Display Properties dialog box.

Help Menu

You use the commands in the **Help** menu to access information about LabWindows/CVI.

Help»Contents

The **Contents** command invokes the LabWindows/CVI Help.

Help»Windows SDK

The **Windows SDK** command invokes online help for the Windows API functions.

Help»Tip of the Day

The **Tip of the Day** command invokes a dialog box containing tips to help you learn about features in the LabWindows/CVI environment.

Help»Web Links

The **Web Links** command has a submenu that contains links to helpful National Instruments Web sites.

Help»About LabWindows/CVI

The **About LabWindows/CVI** command displays a read-only dialog box with information about your LabWindows/CVI session.

4

User Interface Editor Window

A LabWindows/CVI Graphical User Interface (GUI) can consist of panels, command buttons, pull-down menus, graphs, strip charts, knobs, meters, and many other controls and indicators.

You can create your GUI programmatically using function calls, or you can build a GUI in LabWindows/CVI interactively using the User Interface Editor, a drop-and-drag editor with tools for designing, arranging, and customizing user interface objects. With the interactive User Interface Editor, you can build an extensive GUI for your program without writing a single line of code. When you are finished designing your GUI in the User Interface Editor, you save the GUI as a User Interface Resource (`.uir`) file. This section tells you how to create a GUI interactively. It describes the User Interface Editor and procedures for creating and editing panels, controls, and menu bars.

When you use the User Interface Editor, you create and modify user interface resource (`.uir`) files. Enter the User Interface Editor by selecting **New** or **Open** from the **File** menu and choosing the **User Interface** (`*.uir`) menu item.

User Interface Editor Overview

From the User Interface Editor Window, you can create and edit GUI panels, controls, and menu bars. Use the tool bar beneath the menu bar for high-level editing with the mouse. When you click on a particular tool, the mouse cursor changes to reflect the new editing mode.

You can use the following icons in the User Interface Editor window.

Use the operating tool to operate objects. When you are in the operate mode, events display on the right side of the tool bar. These event displays have a built-in delay to give you time to see each event.

Use the editing tool to select, position, and size objects.

Use the labeling tool to modify text associated with objects.

Use the coloring tool to color objects. Clicking the right mouse button displays a color palette from which you can choose a color. Clicking the left mouse button automatically colors the object with the last color selected in the color palette.

Holding down the <Ctrl> key changes the coloring tool to an eyedropper tool. When you click on an object with the eyedropper tool, the current color of the tool becomes the color of that object. Then you can apply that object's color to another object.

Using the Pop-Up Menus of the User Interface Editor

You can open a pop-up menu by right-clicking on the User Interface Editor window. The type of pop-up menu that appears depends on the surface you click on. The following is a list of the areas in the User Interface Editor that bring up pop-up menus.

- If you click on the User Interface Editor window background, a pop-up menu appears containing commands to create a panel or a menu bar.
- If you click on a panel background, a pop-up menu appears with each of the control types you can create.
- If you click on a control, a pop-up menu appears with commands to generate or view the callback function for the control.

CodeBuilder Overview

With the LabWindows/CVI CodeBuilder, you can create automatically complete C code that compiles and runs based on a user interface (`.uir`) file you are creating or editing. By choosing certain options in the **Code** menu, you can produce skeleton code. Skeleton code is syntactically and programmatically correct code that compiles and runs before you have typed a single line of code. With the CodeBuilder feature, you save the time of typing in standard code included in every program, eliminate syntax and typing errors, and maintain an organized source code file with a consistent programming style. Because a CodeBuilder program compiles and runs immediately, you can develop and test the project you create, concentrating on one function at a time.

When you choose **Code»Generate»All Code**, LabWindows/CVI places the `#include` statements, variable declarations, callback function skeletons, and `main` function in the source code file you specify as the target file. Each function skeleton contains a switch construct with a case statement for every default event you specify. You can set default events for control callback functions and panel callback functions by choosing **Code»Preferences**. Although skeleton code runs, you must customize it to implement the actions you want to take place for each event.

When you generate code for a specific control or panel callback function, LabWindows/CVI places the skeleton code for that function in the target file in the same complete format used for the **Code»Generate»All Code** command. However, this code might not run. In order for a project to run, a `main` function must exist. If you lack the `main` function or any of the callback functions you reference in the `.uir` file, the code is incomplete.

It is good practice to use the **Code»Generate»All Code** option first to produce a running project from the current state of the `.uir` file. Then, after adding panels, controls, or menu items to the `.uir` file, select **Code»Generate»Panel Callback**, **Control Callbacks**, or **Menu Callbacks** to make corresponding additions to the source file.

Also with CodeBuilder, you can make sure that your automatically generated program terminates properly. For a CodeBuilder program to terminate successfully, you must include a call to `QuitUserInterface`. When you choose **Code»Generate»All Code**, the Generate All Code dialog box prompts you to choose which callback functions terminate the program. You can select one or more callback functions to ensure proper program termination.

File Menu

This section describes how to use the commands in the User Interface Editor window **File** menu.

File»New, Open, Save, and Exit LabWindows/CVI

The **New**, **Open**, **Save**, and **Exit LabWindows/CVI** commands in the **File** menu of the User Interface Editor work like **New**, **Open**, **Save**, and **Exit LabWindows/CVI** commands in the Project window. For more information on these commands, refer to the *File Menu* section in Chapter 3, *Project Window*.

File»Save As

Use the **Save As** command to write the `.uir` file to disk using a name you specify. The **Save As** command changes the name on the User Interface Editor window title bar to the new name you specified. If you want to append an extension other than `.uir`, type it in after the filename. If you do not want to append any extension, enter a period after the filename.

File»Save Copy As

The **Save Copy As** command writes the contents of the active window to disk using a name you specify without changing the name of the active window.

File»Close

The **Close** command closes the active window. If you have modified the contents of the window since the last save, LabWindows/CVI prompts you to save the file to disk.

File»Save All

The **Save All** command saves all open files to disk.

File»Add File to Project

The **Add File to Project** command adds the `.uir` file in the current window to the project list.

File»Read Only

The **Read Only** command suppresses the editing capabilities in the current window. When you initially open a file, the **Read Only** command is disabled unless the file is read-only on disk.

File»Print

The **Print** command opens the Print dialog box, which allows you to send the entire `.uir` file or the visible screen area to a printer or a file. The Print dialog box also allows you to set print preferences. The print preferences correspond to the *Print Attributes* topic in the **LabWindows/CVI Online Help**.

Edit Menu

This section explains how to use the commands in the User Interface Editor window **Edit** menu. The **Edit** menu is used for editing panels, controls, and menu bars.

Note **Undo** and **Redo** are enabled when you perform an edit action. **Cut** and **Copy** are enabled when you select a control. **Paste** is enabled when you place an object on the Clipboard using the **Cut** or **Copy** command. If you select an edit command while it is disabled, nothing happens.

Edit»Undo and Redo

The **Undo** command reverses your last edit action, and the screen returns to its previous state. Edit actions are stored on a stack so that you can undo a series of your edit actions. The stack can store up to 100 edit actions. You set the size of the undo stack by selecting **Options»Preferences**.

The **Redo** command reverses your last **Undo** command, restoring the screen to its previous state. **Redo** is helpful when you use the **Undo** command to reverse a series of your edit actions and accidentally go too far. The **Redo** command is enabled only when your previous action was the **Undo** command. Any action disables the **Redo** command.

Actions that you can undo and redo appear dynamically in the menu. For example, when you move a control, the menu presents the option **Undo Move Control**.

Edit»Cut and Copy

To cut or copy controls to the Windows Clipboard, select the control you want to place on the Clipboard and then select **Cut** or **Copy** from the **Edit** menu. LabWindows/CVI places the selected control on the Clipboard. If you used the **Copy** command, the control remains in the window. Use the Cut command to delete controls from the window. Controls you cut or copy do not accumulate on the Clipboard. Every time you cut or copy a control it replaces the previous contents of the Clipboard.

To use the **Cut** or **Copy** commands, follow these steps:

1. Select the control you want to place on the Clipboard by clicking on the control or pressing <Tab> until the control is highlighted. Select multiple controls by dragging the mouse over the controls. You also can press <Shift-Click> to select multiple controls.
2. Select **Cut** or **Copy** from the **Edit** menu.

Edit»Paste

The **Paste** command inserts controls, panels, or text from the Clipboard. You can paste an object from the Clipboard as many times as you like. Controls or panels remain on the Clipboard until you use **Cut**, **Cut Panel**, **Copy**, or **Copy Panel** again. The **New** and **Open** commands do not erase the Clipboard.

Edit»Delete

The **Delete** command deletes selected controls without placing the controls on the Clipboard. Because **Delete** does not place controls on the Clipboard, you cannot restore deleted controls using the **Paste** command.

Edit»Copy Panel and Cut Panel

The **Copy Panel** and **Cut Panel** commands put an entire panel on the Clipboard. The **Copy Panel** command copies the selected panel and places it on the Clipboard, leaving the selected panel in its original location. The **Cut Panel** command removes the selected panel and places it on the Clipboard. Panels you cut or copy do not accumulate on the Clipboard. Every time you cut or copy a panel it replaces the previous contents of the Clipboard.

To use the **Copy Panel** or **Cut Panel** commands, follow these steps:

1. Select the panel you want to place on the Clipboard by clicking on the panel or pressing <Shift-Ctrl> and the left or right arrow key until the panel is highlighted.
2. Select **Edit»Copy Panel** or **Cut Panel**.

Edit»Menu Bars

The **Menu Bars** command opens the Menu Bar List dialog box.

The list contains all of the menu bars in the resource file, listed by constant prefix. The Menu Bar List dialog box has the following options:

- **Create**—Opens a new Edit Menu Bar dialog box, as seen in Edit Menu Bar dialog box. After you create a menu bar, it appears below the currently selected menu bar in the menu bar list.
- **Edit**—Opens the Edit Menu Bar dialog box for the selected menu bar.
- **Cut**—Deletes the currently highlighted item in the menu bar list and copies it to the menu bar Clipboard.
- **Copy**—Copies the currently highlighted item in the menu bar list to the menu bar Clipboard.
- **Paste**—Inserts the contents of the menu bar Clipboard to the menu bar list. When you use the **Paste** button, the menu bar is inserted above the currently highlighted item in the menu bar list.
- **Done**—Closes the Menu Bar List dialog box.

The Edit Menu Bar dialog box appears when you click on the **Create** or **Edit** buttons on the Menu Bar List.

The Edit Menu Bar dialog box has the following options:

- **Menu Bar Constant Prefix**—Sets the resource ID for the menu bar. You pass this resource ID to `LoadMenuBar` to load the menu bar into memory. The menu bar constant prefix is defined in the `.h` file that LabWindows/CVI generates when you save the `.uir` file. If you do not assign a menu bar constant prefix, the User Interface Editor assigns one for you when you save the `.uir` file.
- **Item**—Sets the name of the current menu, submenu, or menu command. If you type a double underscore before any letter in the **Item** field, the letter appears underlined in the label. The user can select the menu item by pressing <Alt> and that letter.
- **Constant Name**—Sets the constant name of the item, which is appended to the menu bar constant prefix to from the ID for the current item. You pass the ID to functions such as `GetMenuBarAttribute` and `SetMenuBarAttribute`. `GetUserEvent` returns the ID when the current menu item generates a commit event.
- **Callback Function**—This field is optional. In this box, you can type the name of the function to be called when the current menu item generates an event.
- **Modifier Key**—Identifies the keys that users can press to cause the current menu item to execute.
- **Shortcut Key**—Identifies the keys that users can press to cause the current menu item to execute.

- **Dimmed**—Specifies whether the menu item is initially dimmed.
- **Checked**—Specifies whether the menu item initially has a checkmark.
- **Insert New Item**—Inserts a new item above or below the currently selected menu item.
- **Insert Separator**—Inserts a separator above or below the currently selected menu item.
- The left hierarchy button moves the currently selected menu item up one level in the submenu hierarchy.
- The right hierarchy button moves the currently selected item down one level in the submenu hierarchy.
- **View**—Displays the current state of the menu bar and pull-down menus.
- **Cut**—Deletes the currently selected menu item and copies it to the menu Clipboard.
- **Copy**—Copies the currently selected menu item to the menu Clipboard.
- **Paste**—Inserts the menu item currently on the menu Clipboard above the currently selected menu item.
- **OK**—Accepts the current inputs and closes the dialog box.
- **Cancel**—Cancels the operation and removes the dialog box.

Edit»Panel

The **Panel** command opens the Edit Panel dialog box, where you can set the following options:

- The **Source Code Connection** section of the Edit Panel dialog box has the following options:
  - **Constant Name**—Sets the resource ID for the panel. You pass this resource ID to `LoadPanel` to load the panel into memory. The constant name is defined in the `.h` file that LabWindows/CVI generates when you save the `.uir` file. If you do not assign a constant name, the User Interface Editor assigns a constant name when you save the `.uir` file.
  - **Callback Function**—Specifies the name of the function to be called when an event is generated on the panel. Naming a callback function is optional.
- The **Panel Settings** section of the Edit Panel dialog box has the following options:
  - **Panel Title**—Sets the title of the panel.
  - **Menu Bar**—Sets the name of the menu bar.
  - **Close Control**—Designates which control on your panel causes the panel to close.
  - **Top**—Sets the barrier for the top edge of the panel, in pixels.
  - **Left**—Sets the barrier for the left edge of the panel, in pixels.
  - **Height**—Sets the barrier for the height of the panel, in pixels.
  - **Width**—Sets the barrier for the width of the panel, in pixels.

  - **Scroll Bars**—Enables or disables scroll bars on the panel.
  - **Auto-Center Vertically (when loaded)**—Automatically centers the panel in the vertical center of your monitor.
  - **Auto-Center Horizontally (when loaded)**—Automatically centers the panel in the horizontal center of your monitor.
  - **Other Attributes**—Sets behavior and feature attributes of the panel.

    Refer to the panel attributes discussion for more details.

- From the Quick Edit Window, you can perform high-level edits on the panel. The tools in the toolbar operate like the tools in the main User Interface Editor window. Refer to the *User Interface Editor Overview* section for more information.
- The **Attributes for Child Panels** section of the Edit Panel dialog box has the following options:
  - **Frame Style**—Sets the styles of the frame in the Child Panel.
  - **Frame Thickness**—Sets the frame thickness in the Child Panel.
  - **Title Bar Thickness**—Sets the title bar thickness in the Child Panel.
  - **Size Title Bar Height to Font**—Sets the size of the title bar to match the height of the font in the title bar.
  - **Title Style**—Sets the style of the title in the Child Panel.

Edit»Control

The **Control** command opens a dialog box where you can edit a control you have selected. You also can double-click on a control to open this dialog box. The dialog box can have various sections including, **Source Code Connection**, **Control Settings**, **Control Appearance**, **Quick Edit Window**, and **Label Appearance**. The sections available in the dialog box for a selected control vary slightly depending on the type of control that you are editing.

- The **Source Code Connection** section of the edit control dialog box has the following options:
  - **Constant Name**—The User Interface Editor appends the constant name to the panel resource ID to form the ID for the control. The ID identifies the control in any control-specific functions, such as `GetCtrlVal` and `SetCtrlAttribute`. The ID is defined in the `.h` file that LabWindows/CVI generates when you save the `.uir` file. If you do not assign a constant name, the User Interface Editor assigns one for you when you save the `.uir` file.
  - **Callback Function**—In this box you can type the name of the function to be called when an event is generated on the control. Naming a callback function is optional.

- The **Control Settings** section of the Edit control dialog box displays specific attributes for the type of control that you are editing. It contains the data-specific attributes for the control.

  Ring controls and list boxes have a **Label/Value Pairs** button in the Control Settings section of the Edit control dialog box. This button activates the Edit Label/Value Pairs dialog box.
- The **Control Appearance** section of the Edit control dialog box for a numeric control displays specific attributes for the type of control that you are editing. It contains attributes pertaining to the physical appearance of the control.
- From the Quick Edit Window, you can perform high-level edits on the control. The tools in the tool bar operate like the tools in the main User Interface Editor window. The Quick Edit Window also immediately reflects any changes you make in other sections of the dialog box.

  The Edit control dialog box of any control allows you to interactively set all of the attributes of the control. The control types discussion describes these attributes in detail.
- The **Label Appearance** section of the Edit control dialog box contains attributes pertaining to the physical appearance of the control label.

  If you type a double underscore before any letter in the **Label** field, the letter appears underlined in the label. The user can select the control by pressing <Alt> and the underlined letter, provided that no accessible menu bars contain a menu with the same underlined letter.

Edit»Tab Order

Each control on a panel has a position in the tab order. The tab order determines which control becomes the next active control when the user presses <Tab> or <Shift-Tab>.

When you create a control, it positions itself at the end of the tab order. When you copy and paste a control, the tab position of the pasted control is immediately before the control you copied. Select **Edit»Tab Order** to display the Edit Tabbing Order dialog box.

Click on a control with the pointer cursor to change the tab position of a control to the number in the **Click To Set To** box.

You can change the cursor to the eyedropper cursor by holding down the <Ctrl> key. Click on a control with the eyedropper cursor to change the number in the **Click To Set To** box to the current tab position associated with the control.

Click **OK** button to accept the new tab order.

Click the close button to erase the new tab order and restore the original tab order. For each control, the original tab order appears in dim display to the right of the new tab order you enter.

Edit»Set Default Font

Select **Edit»Set Default Font** to make the font of the currently selected control the default control font. If the label is also selected or is the only item selected, the font of the label becomes the default label font. Newly created controls inherit the default fonts.

Edit»Apply Default Font

Select **Edit»Apply Default Font** to set the font of the currently selected control (and/or label) to the default control font (and/or default label font).

Edit»Control Style

Use the **Control Style** command to change the style of the selected control. For example, you can change a ring slide control to a ring knob control, and the label/value pairs remain intact.

Edit»Edit Custom Controls

You can create custom controls to save control configurations. To create a custom control, create a control as you normally would. Select **Create»Custom Controls»Edit** to display the Edit Custom Controls dialog box. When you click Add in the Edit Custom Control dialog box, you see a list box that contains a list of all the controls on the user interface panel. Select the control you want to add as a custom control, and click OK. The control you added appears in the **Create»Custom Controls** submenu. LabWindows/CVI saves your list of custom controls between sessions. You can rename, delete, and move your custom controls in the Edit Custom Controls dialog box.

Create Menu

This section explains how to use the commands in the User Interface Editor window **Create** menu. Use the **Create** menu to create panels, menu bars, and controls.

Create»Panel

Refer to the *Edit Menu* section for more information on editing the panel.

Create»Menu Bar

Refer to the *Edit Menu* section for more information on editing the menu bar.

Create»Controls

The remaining options in the **Create** menu allow you to create GUI controls. After you create a control, you can modify it using the items in the **Edit** menu. Refer to the *Edit Menu* section for more information on editing controls.

View Menu

This section explains how to use the commands in the User Interface Editor window **View** menu.

View»Find UIR Objects

Use the **Find UIR Objects** command to locate objects in user interface resource (`.uir`) files. When you select this command, the Find UIR Objects dialog box opens.

Select the type or types of objects you want to search for by checking the appropriate checkboxes in the left column of the dialog box.

- **Search By**—Select the search criterion from the **Search By** ring control. The choices available are the following:
  - **Constant Prefix**—Valid for panels and menu bars
  - **Constant Name**—Valid for controls, menus, and menu items
  - **Prefix + Constant Name**—Valid for all
  - **Callback Function Name**—Valid for all, except menu bars
  - **Label**—Valid for all, except menu bars

  Enter the text you want to search for into the string control. You can view a list of all the strings in the file that match the current **Search By** criterion by clicking on the arrow to the right of the string control or by using the up and down arrow keys.
- **Find**—Allows you to select which types of UIR objects to search for.
- **Wrap**—Continues your search at the beginning of the file after reaching the end of the file.
- **Case Sensitive**—Finds instances only of the specified text that match exactly.
- **Whole Word**—Finds the specified text only when it is surrounded by spaces, punctuation marks, or other characters not part of a word. LabWindows/CVI treats the characters A through Z, 0 through 9, and underscore (_) as parts of a word.
- **Regular Expression**—Causes LabWindows/CVI to treat certain characters in the search string control as regular expression characters instead of literal characters. Refer to Table 5-1, , in Chapter 5, *Source and Interactive Execution Windows*, for more information.
- **Find**—Click on the **Find** button to perform the search. If any user interface objects match, a different dialog box appears.

This dialog box allows you to browse through the list of matches. As you come to each object, its callback function name and label appear, and the object is highlighted in the `.uir` file. The Find UIR Objects dialog box has the following buttons:

- **Find Prev**—Searches backward for the previously matched object.
- **Find Next**—Searches forward for the next matching object.
- **Edit**—Terminates the search and opens the Edit dialog box for the user interface object currently highlighted.
- **Stop**—Terminates the search.

View»Show/Hide Panels

The **Show/Hide Panels** command has a submenu.

Use this submenu to **Show All Panels**, **Hide All Panels**, or select individual panels you want to view in the User Interface Editor window.

View»Bring Panel to Front

The **Bring Panel to Front** command has a submenu that lists all panels and allows you to select a panel to bring to the front for editing.

View»Next Panel

The **Next Panel** command brings the next panel in the current `.uir` file to the front for viewing and editing.

View»Previous Panel

The **Previous Panel** command brings the previous panel in the current `.uir` file to the front for viewing and editing.

View»Preview User Interface Header File

The **Preview User Interface Header File** command opens a Source window with a preview of the header file that LabWindows/CVI generates when you save the `.uir` file in the User Interface Editor window.

Arrange Menu

This section explains how to use commands in the **Arrange** menu of the User Interface Editor window.

Arrange»Alignment

The **Alignment** command allows you to align controls on a panel. You can use the mouse to select a group of controls by dragging over them or <Shift-Click> on each item you want to include in the group. Then you can select an alignment method from the submenu. The options on the **Alignment** command submenu are as follows:

Left Edges vertically aligns the left edges of the selected controls to the left-most control.

Horizontal Centers vertically aligns the selected controls through their horizontal centers.

Right Edges vertically aligns the right edges of the selected controls to the right-most control.

Top Edges horizontally aligns the top edges of the selected controls to the upper-most control.

Vertical Centers horizontally aligns the selected controls through their vertical centers.

Bottom Edges horizontally aligns the bottom edges of the selected controls to the lower-most control.

Arrange»Align

The **Align** command performs the same action as the **Alignment** command, using the option you last selected in the **Alignment** command submenu.

Arrange»Distribution

The **Distribution** command allows you to distribute controls on a panel. Select a group of controls by dragging the mouse over them or <Shift-Click> on each item you want to include in the group. Then you can select a distribution method from the submenu. The options on the **Distribution** command submenu are as follows:

Top Edges sets equal vertical spacing between the top edges of the controls. The upper-most and lower-most controls serve as anchor points.

Vertical Centers sets equal vertical spacing between the centers of the controls. The upper-most and lower-most controls serve as anchor points.

Bottom Edges sets equal vertical spacing between the bottom edges of the controls. The upper-most and lower-most controls serve as anchor points.

Vertical Gap sets equal vertical gap spacing between the controls. The upper-most and lower-most controls serve as anchor points.

Vertical Compress compresses the spacing of controls to remove any vertical gap between the controls.

Left Edges sets equal horizontal spacing between the left edges of the controls. The left-most and right-most controls serve as anchor points.

Horizontal Centers sets equal horizontal spacing between the centers of the controls. The left-most and right-most controls serve as anchor points.

Right Edges sets equal horizontal spacing between the right edges of the controls. The left-most and right-most controls serve as anchor points.

Horizontal Gap sets equal horizontal gap spacing between the controls. The left-most and right-most controls serve as anchor points.

Horizontal Compress compresses spacing of the controls to remove any horizontal gap between the controls.

Arrange»Distribute

The **Distribute** command performs the same action as the **Distribution** command, using the option you last selected in the **Distribution** command submenu.

Arrange»Control ZPlane Order

The **Control ZPlane Order** option lets you set the sequence in which overlapped controls are drawn. Controls are always drawn in order, from the back to the front of the z-plane order. The **Control ZPlane Order** submenu presents four commands:

- **Move to Front** moves the control to the front of the z-plane order so it is drawn last.
- **Move to Back** moves the control to the back of the z-plane order so it is drawn first.
- **Move Forward** moves the control one place forward in the z-plane order.
- **Move Backward** moves the control one place backward in the z-plane order.

Arrange»Center Label

The **Center Label** command centers the label of the selected control.

Arrange»Control Coordinates

The **Control Coordinates** command invokes a dialog box where you can interactively set the width, height, top, and bottom of all selected controls and labels.

Code Menu

This section explains how to use the commands in the User Interface Editor window **Code** menu. Use the commands in the **Code** menu to use CodeBuilder to generate code automatically based on a (`.uir`) file you are creating or editing.

Code»Set Target File

Use the **Set Target File** command to set the destination file for the **Insert Function Call** command. **Set Target File** brings up a dialog box from which you can select from any open Source window or the Interactive Execution window.

Code»Generate

You access the CodeBuilder features of LabWindows/CVI in the **Generate** menu item. The commands in the **Generate** menu produce code based on the `.uir` file. The code produced by the **Generate** menu uses the bracket styles you specify with the **Bracket Styles** command in the **Options** menu of the Source window for your project.

The **Generate** command has a submenu that contains the following commands:

- **All Code**—Generates code to accompany the `.uir` file.
- **Main Function**—Generates code for the `main` function and writes it to the target file.

- **All Callbacks**—Generates code for all the callback functions and writes them to the target file.
- **Panel Callbacks**—Generates code for the callback function associated with a panel.
- **Control Callbacks**—Generates code for the callback functions associated with one or more controls.
- **Menu Callbacks**—Generates code for menus and menu items connected to callback functions.

The **All Callbacks** command is available when any of the **Panel Callback**, **Control Callbacks**, or **Menu Callbacks** commands are available.

The **Panel Callback** command is available if you specified a callback function for the currently active panel. The **Control Callbacks** command is available if you have specified callback functions for any of the currently selected controls. The **Menu Callbacks** command is available if you have a menu bar that contains items for which you specified a callback.

When you generate code to accompany a `.uir` file, LabWindows/CVI places the skeleton code in the target file. You must save the `.uir` file before you can generate any code based on that file. When you save a `.uir` file, LabWindows/CVI generates a header (`*.h`) file with the same name. This `.h` file and `userint.h` are included in the source file.

If you try to generate the same function more than once, the Generate Code dialog box appears. Each previously generated code fragment appears highlighted. Click on the appropriate button in the Generate Code dialog box to replace the existing function, insert a new function, or skip to the next generated function.

Generate»All Code

Selecting **Code»Generate»All Code** opens the Generate All Code dialog box. This dialog box displays a checklist and prompts you to choose the panel or panels that the `main` function loads and displays at run time. LabWindows/CVI automatically assigns a default panel variable name for each panel in the `.uir` file.

The Generate All Code dialog box also prompts you to choose the callback function or functions that terminate the program. For a CodeBuilder program to terminate successfully, you must include a call to `QuitUserInterface`.

Note Callback functions associated with close controls are automatically checked in the **Program Termination** section of the Generate All Code dialog box. You can define a control to be a close control in the Edit Panel dialog box by selecting **Edit»Panel**.

If you have ActiveX servers in your panel, the Generate All Code dialog box lists the ActiveX servers corresponding to all the controls that you created in the User Interface Editor window.

For each server that you select, LabWindows/CVI runs the ActiveX Controller Wizard when you generate all code.

To automatically generate all code, select the panels you want to load and display in the user interface. Also select the callback function or functions you want to terminate the program and then click on **OK**.

When you choose **Code»Generate»All Code**, LabWindows/CVI produces the `#include` statements, the variable declarations, the function skeletons and the `main` function, and places them in the target file. The callback functions you selected to terminate program execution include a call to the User Interface Library `QuitUserInterface` function.

Unless you have selected the **Code»Preferences»Always Append Code to End** option, LabWindows/CVI places the skeleton code for each callback function at the cursor position in the target file. If the cursor is inside an existing function, LabWindows/CVI repositions the cursor at the end of that function before inserting the new function. CodeBuilder places all functions of one type (panel callback, control callback, or menu callback) together in the source file. Any panel callbacks are placed first in the source file, control callbacks are placed next, and menu callbacks are placed last. Refer to the *Code»Preferences* section for more information on specifying the location of generated code.

Function skeletons for control and panel callbacks include the complete prototype, the proper syntax, a return value, and a switch construct containing a case for each default control or panel event. Function skeletons for menu callbacks include the complete prototype and open and close brackets. You can set the default events by selecting **Code»Preferences**. Refer to the *Code»Preferences* section for more details. You can set the location of the open and close brackets by selecting **Options»Bracket Style** in a Source window.

Generate Main Function

Selecting **Code»Generate»Main Function** option opens the Generate Main Function dialog box. This dialog box prompts you to choose the panels the `main` function loads and displays at run time. LabWindows/CVI automatically assigns a default panel variable name for each panel in the `.uir` file.

Note If you previously selected the **Code»Generate»All Code** command, you do not have to execute this command. Use this command only when you want to replace the main callback function to add or change the panels to be loaded at run time.

To automatically generate code for the `main` function, select the panel or panels you want to load and display in the user interface and then click on **OK**.

When you choose **Code»Generate»Main Function**, LabWindows/CVI produces the `#include` statements, the variable declarations, and the main callback function, and places them in the target file.

Note If the source file contains only the `main` function and the `#include` statements and you have not yet created the appropriate callback functions, you might get an error when trying to run the project. When the main function calls `LoadPanel`, LabWindows/CVI generates a non-fatal error for each callback function it cannot find in the source file.

Note The **Generate WinMain( ) Instead of Main( )** checkbox enables you to use `WinMain` instead of `main` for your main program. In LabWindows/CVI, you can use either function as your program entry point. When linking your application in an external compiler, it is easier to use `WinMain`.

If your project target is a DLL, neither `WinMain` or `main` are generated. Instead, CodeBuilder generates a `DLLMain` function and places the bulk of the User Interface function calls in a function called `InitUIForDLL`. Call `InitUIForDLL` in your DLL at the point you want to load and display panels.

When you link your executable or DLL in an external compiler, you must include a call to `InitCVIRTE` in `WinMain`, `main`, or `DLLMain` (or `DLLEntryPoint` for Borland C/C++). In a DLL, you must also include a call to `CloseCVIRTE`. Refer to Chapter 3, *Compiler/Linker Issues,* in the *Measurement Studio LabWindows/CVI Programmer Reference Manual*. CodeBuilder automatically generates the necessary calls to `InitCVIRTE` and `CloseCVIRTE` in your `WinMain`, `main`, or `DLLMain` function. It also automatically generates a `#include` statement for the `cvirte.h` file.

Generate All Callbacks

When you select **Code»Generate»All Callbacks**, LabWindows/CVI produces the `#include` statements and the callback function skeletons and places them in the target file.

Generate Panel Callbacks

Before you can choose **Code»Generate»Panel Callback**, you must activate a panel.

When you select **Code»Generate»Panel Callback**, LabWindows/CVI produces the `#include` statements and the function skeleton for the active panel and places them in the target file.

Generate Control Callbacks

Before you can choose **Generate»Control Callbacks**, you must select at least one control.

When you select **Code»Generate»Control Callbacks**, LabWindows/CVI produces the `#include` statements and the function skeleton for each selected control and places them in the target file.

You also can generate a control callback function skeleton by clicking on the control with the right mouse button and selecting the **Generate Control Callback** command from the pop-up menu.

Generate Menu Callbacks

Selecting **Code»Generate»Menu Callbacks** opens the Select Menu Bar Objects dialog box. Select the menu bar objects for which you want to generate callbacks and then click on **OK**.

When you select **OK**, LabWindows/CVI produces the `#include` statements, the function prototypes, and the opening and closing brackets for each callback function. No switch construct or case statements are produced because the usual default events do not apply to menu callback functions. You must add the code to implement the actions you want to take place when a menu bar item is selected.

Code»View

Use the **Code»View** command to look at code for a given callback function.

To view the code for a function from the `.uir` file, select a panel or control and then select **View»Panel Callback** or **View»Control Callback**. The source file containing the callback function appears with the function name highlighted. You also can view the code for a control callback function by clicking on the control with the right mouse button and selecting the **View Control Callback** command from the pop-up menu.

When you choose the **View** command for a callback function, LabWindows/CVI searches for that function in all open Source windows, in all the source files in the project, and in any other open source files. If the function is found in a closed project file, that file is opened automatically.

The **View** command is useful because the callback functions for one user interface can be in several different files, and scrolling the source code is not efficient. With the **View** command, you can move instantly from the user interface file to an object callback function, whether the source file is open or closed.

When you are finished reviewing the code, you can return instantly to the `.uir` file from the source file. To return to the `.uir` file, place the cursor on the callback function name or constant name of the User Interface object you want to go to and select the **Find UI Object** command from the **View** menu in the Source window.

Note You cannot use the **View** command for menu callback functions.

Code»Preferences

Use the **Preferences** command to change the default settings of case statements generated for control callback functions and panel callback functions or to specify the target file location for generated code.

Use the **Default Panel Events** or **Default Control Events** commands to select which events LabWindows/CVI places into the switch construct of the code for panel or control callback functions, respectively. You can choose from several events, and you can choose to **Add 'Default:' Switch Case**. Selecting **Code»Preferences»Default Panel Events** opens the Panel Callback Events dialog box. Selecting **Code»Preferences»Default Control Events** opens the Control Callback Events dialog box.

To set the **Default Panel Events** or **Default Control Events**, select the events you want to be included in the code as case statements and then click on **OK**. For each option you choose, LabWindows/CVI includes in the source code a case statement that corresponds to this option.

Note Default control events are ignored for timer control callbacks, for which the only event cases are `EVENT_TIMER_TICK` and `EVENT_DISCARD`.

When the **Always Append Code to End** command is selected, LabWindows/CVI places the skeleton code for each callback function at the end of the target file. When this option is not selected, newly generated code is placed at the current position of the cursor in the target file.

Run Menu

This section explains how to use the commands in the User Interface Editor window **Run** menu. Refer to the *Run Menu* section in Chapter 5, *Source and Interactive Execution Windows*, for a description of the following commands:

- Debug
- Continue
- Step Over
- Step Into
- Finish Function
- Terminate Execution
- Break at First Statement
- Breakpoints

Library Menu

The **Library** menu for the User Interface Editor window works the same way as the **Library** menu in the Project window. Refer to each library overview in the *Library Reference* section of the **LabWindows/CVI Help**.

Tools Menu

The **Tools** menu for the User Interface Editor window works the same way as the **Tools** menu in the Project window. Refer to the *Tools Menu* section in Chapter 3, *Project Window*, for information about the **Tools** menu.

Window Menu

The **Window** menu in User Interface Editor window works the same way as the **Window** menu in the Project window. Refer to the *Window Menu* section in Chapter 3, *Project Window*, for information about the **Window** menu.

Options Menu

This section explains how to use the commands in the User Interface Editor window **Options** menu.

The **Options** menu includes the following commands:

Options»Operate Visible Panels

Operate Visible Panels allows you to operate the visible panels as you would in an application program. This command has the same effect as clicking on the operating tool, shown at left. When you finish operating the panel, select **Operate Visible Panels** again to return to edit mode.

Options»Next Tool

The **Next Tool** command in the **Options** menu cycles the User Interface Editor through its four modes. For more information on the operating, editing, labeling, and coloring tools, refer to the *User Interface Editor Overview* section.

Options»Preferences

Selecting **Options»Preferences** opens the Editor Preferences dialog box.

- Use the **User Interface Editor Preferences** section of the Editor Preferences dialog box to set options that affect the operation of the User Interface Editor.
  - **Initial Editor Background Color**—Determines the initial background color of User Interface Editor windows.
  - **Use Localized Decimal Symbol**—Replaces the traditional decimal symbol used by LabWindows/CVI in numeric, graph, and table controls with the decimal symbol specified in the Regional Settings Properties configuration of your Windows Control Panel.
- Use the **Preferences for New Panels** section of the Editor Preferences dialog box to set initial attribute values for each panel that you create in the User Interface Editor.
  - **Resolution Adjustment(%)**—Specifies the degree to which LabWindows/CVI scales your panels and their contents when you display them on screens with resolutions different than the one on which you create them. This option also appears in the Other Attributes dialog box that you can activate from the Edit Panel dialog box by selecting **Edit»Panel** in the User Interface Editor window.

    To programmatically override this setting, you can call the `SetSystemAttribute` with the `ATTR_RESOLUTION_ADJUSTEMENT` attribute before calling `LoadPanel` or `LoadPanelEx`.
  - **Conform to System Colors**—Forces panels and the controls they contain to use the system colors. This option also appears in the Other Attributes dialog box that you can activate from the Edit Panel dialog box by selecting **Edit»Panel**. To programmatically set this option, you can call SetPanelAttribute with the `ATTR_CONFORM_TO_SYSTEM` attribute. When this option is enabled, you cannot change any panel or control colors.
  - **Use System Colors as Defaults for Panels and Controls**—LabWindows/CVI uses the system colors as the initial colors for panels and controls you create when this box is checked. You can subsequently change the colors without restriction.

Note You *must* disable two options, **Conform to System Colors** and **Use System Colors as Defaults for Panels and Controls**, to set the following options: background color, frame color, and title bar color. The frame color and title bar color options have effect only when you load a panel as a child panel. To change each of these three options in the User Interface Editor, you can use the Paintbrush tool on the background, frame, or title bar of a panel. To set these colors programmatically, use `SetPanelAttribute` with the `ATTR_BACKCOLOR`, `ATTR_FRAME_COLOR`, and `ATTR_TITLE_BACKCOLOR` attributes.

- Use the **Preferences for New Controls** section of the Editor Preferences dialog box to set initial attribute values for each control that you create in the User Interface Editor. The **Control Text Style** and **Label Text Style** command buttons allow you to select the initial font and text style for all new controls.
- Click on the **More** button to open the Other User Interface Editor Preferences dialog box.

Options»Assign Missing Constants

The **Assign Missing Constants** command assigns constant names to all of the objects in the User Interface Editor window that currently do not have constant names. A confirmation dialog box appears showing the number of items that have no constant names.

Options»Save In Text Format

The **Save In Text Format** command saves the contents of the User Interface Editor window in an ASCII text format. A dialog box appears prompting you to enter the pathname under which to save the text file. The extension `.tui` is recommended for such files. Do *not* use the `.uir` extension.

The ASCII text file contains descriptions of all the objects in the User Interface Editor window. You can call `LoadPanel` and `LoadPanelEx` on `.tui` files.

Note If you have a large number of objects in your User Interface Editor window, loading a `.tui` file can take significantly longer than loading a comparable `.uir` file.

Note The `.tui` file format in LabWindows/CVI 5.0 and later differs from previous versions. If you use `.tui` files to find differences between versions of your `.uir` files and you created `.tui` files in previous versions of LabWindows/CVI, create new baseline `.tui` files for your `.uir` files.

Options»Load From Text Format

The **Load From Text Format** command loads into a new User Interface Editor window the objects defined in a file saved using the **Save In Text Format** command. A dialog box appears prompting you for the pathname of the file.

Help Menu

The **Help** menu for the User Interface Editor window works the same way as the **Help** menu in the Project window. Refer to the *Help Menu* section in Chapter 3, *Project Window*, for information about the **Help** menu.

5

Source and Interactive Execution Windows

Source Windows

Source windows display the source code for the programs you develop. These windows behave like standard text editors. You can type text directly into a Source window or load text from an ASCII file into a Source window. You can insert code from LabWindows/CVI function panels directly into Source windows. You can save a program from a Source window as an ASCII file. Source windows can contain up to 1 million lines with up to 1,020 characters in each line. A tab is one character for the purpose of line length limitation.

When you run a program in a Source window, the program must be complete and obey the syntax rules of ANSI C. Refer to the *Build Menu* and *Run Menu* sections for more information on running programs.

Toolbars in LabWindows/CVI

The LabWindows/CVI toolbar appears within function panels, in the Function Panel Editor window, and in Source windows. Using the toolbar gives you quick access to common commands, such as **File»Open** and **File»Save**. You can configure the toolbar to meet your needs or choose not to display it at all.

To find out what a toolbar button does, position the mouse cursor over that button, and either hold the cursor there for a short period of time or right-click on the toolbar button to display the name of the toolbar button.

Modifying Your Toolbars

To modify a toolbar, choose **Options»Toolbar** to display the Customize Source Window Toolbar dialog box.

The list box on the left of the Customize Source Window Toolbar dialog box contains names and icons of toolbar buttons that do not currently appear in the toolbar. The list box on the right contains the names and icons of toolbar buttons that currently appear in the toolbar.

The Customize Source Window Toolbar dialog box gives you several ways to configure your toolbar.

Adding and Positioning Buttons

Use the Add Button controls to add and position new buttons on the toolbar. First, select the button you want to add to the toolbar from the list box on the left. In the list box on the right, select the button you want to place the new button next to. The **Above** button positions the item you are adding above the button that you selected in the list box on the right. After you click on **OK**, the new item appears to the left of the other button in the toolbar. The **Below** button positions the item you are adding below the button that you selected in the list box on the right. After you click on **OK**, the new item appears to the right of the other button in the toolbar.

Adding and Positioning Separators

Use the Add Separator controls to add and position separators on the toolbar. Select a button in the list box on the right. The **Above** button adds a separator above the button that you selected in the list box on the right. After you click on **OK**, a small gap (the separator) appears in the toolbar to the left of the selected button. The **Below** button adds a separator below the button that you selected in the list box on the right. After you click on **OK**, a small gap (the separator) appears in the toolbar to the right of the selected button.

Other Positioning Controls

You can select any item in the list box on the right and use the **Remove** button to remove it from the toolbar. If you remove a button, it moves to the list box on the left. If you remove a separator, it disappears from the list. Your modifications take effect on the toolbar when you click on **OK**.

Click on **Default** to restore the default toolbar configuration for LabWindows/CVI.

To position any item on the toolbar, select it in the list box and click on **Move Up** or **Move Down**. Your modifications take effect on the toolbar when you click on **OK**.

Notification of External Modification

If you have externally modified a file since you last loaded or saved it in LabWindows/CVI and the file is in a Source window, a dialog box appears when you switch back to LabWindows/CVI from another Windows application. You are given the option of updating the Source window from the file on disk, overwriting the file on disk with the contents of the Source window, or doing nothing.

Context Menus

You can open a context menu in the Source window by clicking the right mouse button. The context menu contains a set of the most commonly used menu commands from the Source window menu bar. The set of commands is different depending on whether the mouse is over the text editing area or over the line number or line icon area.

Interactive Execution Window

You can execute selected portions of code in the Interactive Execution window. Unlike the Source window, you do not have to have a complete program in the Interactive Execution window. For instance, you can execute C variable declarations and assignment statements without declaring a `main` function.

Use the Interactive Execution window to test portions of code before you include them in your main program. Also, you can use the Interactive Execution window to execute functions exported by a loaded instrument or by a file in the project if the project has been linked. The Interactive Execution window can access functions and data declared as global in a Source window, but a Source window has no access to the functions and data declared in the Interactive Execution window.

When you execute a function from a function panel, LabWindows/CVI inserts the function call into the Interactive Execution window for execution. In this way, the Interactive Execution window keeps a record of the functions you execute from function panels.

When LabWindows/CVI copies a function call from a function panel to the Interactive Execution window for execution, it inserts the code after all the pre-existing lines. LabWindows/CVI also inserts an include statement for the header file associated with the function in the Interactive Execution window if you have not already included it. When you execute a function call from a function panel, LabWindows/CVI automatically excludes all previous lines in the Interactive Execution window. An excluded line is dimmed and the LabWindows/CVI compiler ignores it. Refer to the *Edit»Toggle Exclusion* section for more information about excluded lines.

When you execute code in the Interactive Execution window, LabWindows/CVI automatically excludes all declarations. This is why you must avoid placing executable statements on the same line as declarations in the Interactive Execution window. Auto-exclusion also occurs when you type a line of code beneath a line that has just been executed. You can manually exclude and include lines with the **Edit»Toggle Exclusion** command.

Declarations in the Interactive Execution window remain in effect until you select **Build»Clear Interactive Declarations** or **Edit»Clear Window**.

Rules for executing code in the Interactive Execution window are as follows:

- When executing code from the Interactive Execution window, data declarations must precede any program statements. Function declarations are also necessary unless you disable the **Require Function Prototypes** option in the Build Options dialog box by selecting **Options»Build Options** in a Project window.
- You cannot include function definitions in the Interactive Execution window. LabWindows/CVI treats the following statements the same:

  ```
  extern int fn (void);
  int fn (void);
  ```

  LabWindows/CVI treats the following statements as errors:

  ```
  static int fn (void);
  static int fn () {}
  ```

- LabWindows/CVI treats all global data declarations in the Interactive Execution window as if they are declared as static unless the `extern` keyword precedes them. If the `extern` keyword precedes them, the global declaration must exist in a loaded instrument or in a file in the project.

  The following data declaration is invalid in the Interactive Execution window:

  ```
  extern int x=6;
  ```

Using Subwindows

The Source and Interactive Execution windows support subwindows so that you can have two scrollable editing areas for the same file. To create a subwindow from any of these windows, use the mouse to drag the thin line beneath the menu bar (or toolbar if you have activated the **Toolbar** option from the **View** menu) to a lower position in the window. You can then switch between the subwindows by pressing <F6> or by clicking in a subwindow with the mouse.

Selecting Text in the Source and Interactive Execution Window

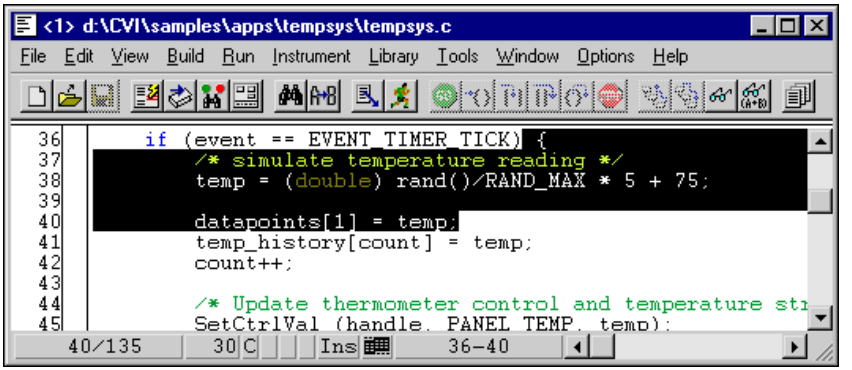
Certain LabWindows/CVI commands require that you select the block of text to which the next command applies. In LabWindows/CVI, you can select a range of characters, a range of lines, or a range of columns. When you select a block of text, it is highlighted on the screen.

To select text with the keyboard, hold down the <Shift> key as you move the keyboard cursor over the text you want to select. You can use the <Shift> key in combination with any of the keyboard commands for moving the keyboard cursor or scrolling the window.

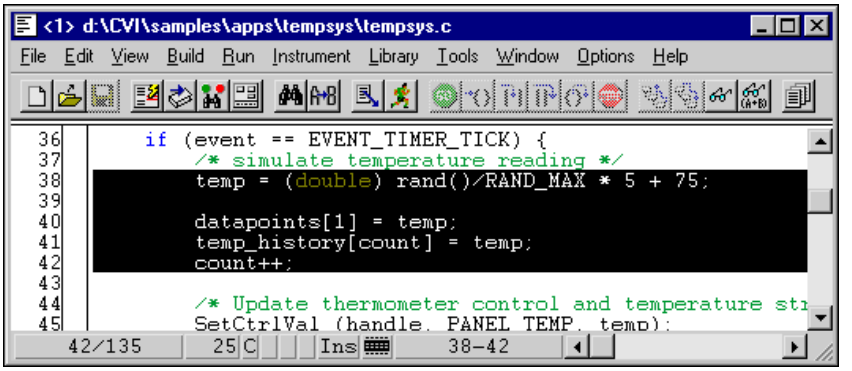
To select text with the mouse, click on the first character you want to select and drag the mouse over the remaining characters. To select a word, double-click on the word. To select a line, triple-click on the line. If you make a mistake while selecting text, click the mouse or press <Esc> to cancel the selection.

LabWindows/CVI provides three modes for selecting text depending on the state of the graphical icon at the bottom of the window, as illustrated in the following figures.

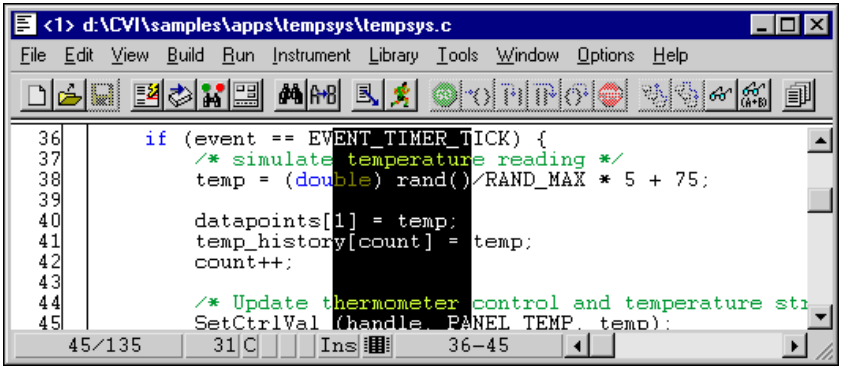
Character Select mode highlights all characters from where you begin selecting text to where you end the selection.


<1> d:\CVI\samples\apps\tempsys\tempsys.c
File Edit View Build Run Instrument Library Tools Window Options Help
36 if (event == EVENT_TIMER_TICK) {
37 /* simulate temperature reading */
38 temp = (double) rand()/RAND_MAX * 5 + 75;
39
40 datapoints[1] = temp;
41 temp_history[count] = temp;
42 count++;
43
44 /* Update thermometer control and temperature str
45 SetCtrlVal (handle, PANEL_TEMP, temp);
40/135 30 C Ins 36-40

Line Select Mode highlights full lines of text.


<1> d:\CVI\samples\apps\tempsys\tempsys.c
File Edit View Build Run Instrument Library Tools Window Options Help
36 if (event == EVENT_TIMER_TICK) {
37 /* simulate temperature reading */
38 temp = (double) rand()/RAND_MAX * 5 + 75;
39
40 datapoints[1] = temp;
41 temp_history[count] = temp;
42 count++;
43
44 /* Update thermometer control and temperature str
45 SetCtrlVal (handle, PANEL_TEMP, temp);
42/135 25 C Ins 38-42

Column Select Mode highlights a rectangular block of text.


<1> d:\CVI\samples\apps\tempsys\tempsys.c
File Edit View Build Run Instrument Library Tools Window Options Help
36 if (event == EVENT_TIMER_TICK) {
37 /* simulate temperature reading */
38 temp = (double) rand()/RAND_MAX * 5 + 75;
39
40 datapoints[1] = temp;
41 temp_history[count] = temp;
42 count++;
43
44 /* Update thermometer control and temperature str
45 SetCtrlVal (handle, PANEL_TEMP, temp);
45/135 31 C Ins 36-45

You can cycle through these three modes by pressing <Ctrl-Insert> on the keyboard or by clicking the mouse on the graphical icon at the bottom of the window.

File Menu

This section explains how to use the commands in a Source and Interactive Execution window **File** menu.

File»New, Open, Save, and Exit LabWindows/CVI

The **New**, **Open**, **Save**, and **Exit LabWindows/CVI** commands in the **File** menu of the User Interface Editor work like **New**, **Open**, **Save**, and **Exit LabWindows/CVI** commands in the Project window. For more information on these commands, refer to the *File Menu* section of Chapter 3, *Project Window*.

File»Open Quoted Text

The **Open Quoted Text** command opens `.c`, `.h`, `.fp`, and `.uir` files that appear by name in the active window. If you select **Open Quoted Text** when the text cursor in the active window is on a line that contains a filename in quotation marks or angle brackets, that file opens in the corresponding window type.

File»Save As

The **Save As** command writes the contents of the active window to disk using a new filename you specify and changes the name of the active window to the name you specified.

File»Save Copy As

The **Save Copy As** command writes the contents of the active window to disk using a filename you specify *without* changing the name of the active window.

File»Close

The **Close** command closes the active window. If you have modified the contents of the window since the last save, LabWindows/CVI prompts you to save the file to disk.

Note The **Hide** command replaces the **Close** command in the Interactive Execution window. The **Hide** command visually closes the Interactive Execution window but retains their contents in memory.

File»Save All

The **Save All** command saves all open files to disk.

File»Add File to Project

The **Add File to Project** command adds the file in the current window to the project list.

File»Read Only

The **Read Only** command suppresses the text editing capabilities in the current window. When you initially open a file, LabWindows/CVI disables the **Read Only** command unless the file is read only on disk.

File»Print

The **Print** command prints the window contents to a printer or a file.

Edit Menu

This section explains how to use the commands in a Source or Interactive Execution window **Edit** menu, as shown in Edit Menu. Use the commands in the **Edit** menu to edit text in Source windows and the Interactive Execution window.

Note Selecting Text in the Source and Interactive Execution Window describes the procedures for moving the cursor, scrolling, and selecting text.

Edit»Undo and Redo

Note LabWindows/CVI disables **Undo** and **Redo** until you make an edit. LabWindows/CVI disables the **Cut** and **Copy** commands until you select text and disables **Paste** until you place text on the Clipboard. If you select an edit command while it is disabled, nothing happens.

The **Undo** command reverses your last edit action. LabWindows/CVI stores editing actions in a stack so that sequential **Undo** commands reverse a history of your edit actions. You can set the size of this stack using the **Options»Editor Preferences** command. The maximum capacity of this stack is 1,000 operations.

The **Redo** command reverses your last **Undo** command. If you go too far in using the **Undo** command, you can use **Redo** to reverse your edit actions. LabWindows/CVI enables the **Redo** command only when the previous action was the **Undo** command. Any other action, even moving the cursor, disables the **Redo** command.

Edit»Cut and Copy

To cut or copy text to the Windows Clipboard, select the text you want to place on the Clipboard and then select **Cut** or **Copy** from the **Edit** menu. LabWindows/CVI places the selected text on the Clipboard. If you used the **Copy** command, the text remains in the window. Use the **Cut** command to delete text from the window. Controls you cut or copy do not accumulate on the Clipboard. Every time you cut or copy a control it replaces the previous contents of the Clipboard.

Edit»Paste

The Paste command inserts text from the Clipboard.

- If you **Paste** in character-select mode, the characters appear at the cursor on the current line.
- If you **Paste** in line-select mode, the new lines appear above the current line.
- If you **Paste** in column-select mode, the new block of characters appears at the cursor on the current line.
- If you select text before you execute the **Paste** command, the contents of the Clipboard replace the selected text.

You can **Paste** the same information from the Clipboard as many times as you like. Text remains on the Clipboard until you use **Cut** or **Copy** again or until another application overwrites the Clipboard. The **New** and **Open** commands do not erase the Clipboard.

To insert text from the Clipboard, move the cursor to the place you want the text inserted and select **Paste** from the **Edit** menu.

Edit»Delete

The **Delete** command deletes highlighted text without placing the text on the Clipboard.

Edit»Select All

The **Select All** command selects all the text in the Source window and positions the keyboard cursor at the end of the file.

Edit»Clear Window

Use the **Clear Window** command in the Interactive Execution window to clear the contents of the window. The **Clear Window** command also clears any variables declared in the Interactive Execution window.

Note The **Clear Window** command is disabled in Source windows.

Edit»Toggle Exclusion

You can specify portions of code to exclude during compilation and execution. LabWindows/CVI ignores excluded code and displays it in a different color than included code.

The **Toggle Exclusion** command marks lines in Source windows and the Interactive Execution window as excluded or included code. This command acts on single and multiple line selections.

You can exclude lines automatically when working in the Interactive Execution window. Refer to *Interactive Execution Window* for more information on automatically excluding lines. Select **Edit»Toggle Exclusion** if you want to include these lines.

Edit»Resolve All Excluded Lines

The **Resolve All Excluded Lines** command interactively highlights the next excluded line or set of consecutive excluded lines and allows you to reinclude, comment out, conditionally compile out, delete, or skip the code.

Edit»Insert Construct

The **Insert Construct** command has a submenu of various C programming constructs. Use this command to insert a construct into your Source window at the current keyboard cursor position.

For most of these menu items, a dialog box appears asking you to fill in portions of the construct. You can press <Enter> or click on **OK** without filling in the controls.

When you insert the construct into your program, the keyboard cursor moves to the first location in the construct where you can enter text.

You can set the location of the curly brackets in the construct using the **Options»Bracket Styles** command.

Edit»Balance

Use the **Balance** command to find pairs of opening and closing curly braces, brackets, and parentheses. If the cursor is within (or near) a set of any of these symbols when you select the **Balance** command, LabWindows/CVI highlights all characters between them. This command is useful when you want to find a missing opening or closing symbol and a large number of these symbols are nested inside each other.

Edit»Diff

Use the **Diff** command for comparing two source files to detect any differences.

The **Diff** command has a submenu that you can use to do the following:

- Use **Diff With** from a Source window to select another open source file and compare it against the current source file.
- After you select the two files to compare, use **Synchronize at Top** to display both files starting at the top.
- Select **Find Next Difference** to display the next point where a difference exists in the files.
- Highlight a section from one of the files and select **Synchronize Selections** from that window to find a matching section in the other file.
- Use **Match Criteria** to establish the number of lines that must match to mark the end of differing sections in a file.
- Select **Ignore White Space** to compare files while ignoring spaces, tabs, or other text control characters.
- Use **Recompare Ignoring White Space** once a difference has been found to determine if the only difference in the selections involves white space characters.

Edit»Go To Definition

When you place the text cursor on a C identifier and select **Go To Definition**, LabWindows/CVI highlights the definition of the identifier. If the definition is not available, for example, a LabWindows/CVI library function definition, LabWindows/CVI highlights the declaration of the identifier.

Edit»Go To Next Reference

When you place the text cursor on a C identifier and select **Go To Next Reference**, LabWindows/CVI highlights the next reference of the identifier.

Edit»Go Back

Returns to the previous identifier.

Edit»Find

The **Find** command invokes the Find dialog box.

Enter the text you want to find in the **Find What** text box. If you select text on a single line before you execute the **Find** command, the selected text appears in the **Find What** text box. Otherwise, the text you last searched for appears in the box. You can access a history of selections for the **Find What** text box by clicking the mouse on the arrow to the right of the **Find What** text box or by using the up or down arrow keys on your keyboard.

- **Case Sensitive**—Finds only the instances of the specified text that match exactly. For example, if `CHR` is the specified text, the **Case Sensitive** option finds `CHR` but not `Chr`.
- **Whole Word**—Finds the specified text only when the characters that surround it are spaces, punctuation marks, or other characters not considered parts of a word. LabWindows/CVI treats the characters A through Z, a through z, 0 through 9, and underscore (_) as parts of a word.
- **Wrap**—Specifies to continue searching from the beginning of the window once the end of the window has been reached.
- **Regular Expression**—If you select this option, LabWindows/CVI treats certain characters in the **Find What** text box as regular expression characters instead of literal characters. Table 5-1 describes the regular expression characters.

Table 5-1. Regular Expression Characters

Purpose	Character	Description	Example
Wildcard matching	. (period)	Match 1 character	a.t matches act and apt but not abort
Matching zero or more occurrences	* (asterisk)	Match 0 or more occurrences of preceding character or expression	0*1 matches 1, 01, 001, etc. a.* matches act, apt, and abort
	+ (plus sign)	Match 1 or more occurrences of preceding character or expression	0+1 matches 01, 001, 0001, . . .
Matching either/or	? (question mark)	Match 0 or 1 occurrences of preceding character or expression	0?1 matches 1 and 01 but not 001
	| (pipe)	Match either the preceding or following character or expression	a|b matches every occurrence of a or b abor|ut matches every occurrence of abort or about {if}|{else} matches every occurrence of if or else
Matching the beginning or ending of a line	^ (caret)	Match the beginning of a line	^int matches any line that begins with int
	$ (dollar sign)	Match the end of a line	end$ matches any line that ends with end
Grouping expressions	{} (curly braces)	Group characters or expressions for searches	{if}|{else} matches every occurrence of if or else

Table 5-1. Regular Expression Characters (Continued)

Purpose	Character	Description	Example
Matching a set	`[]` (brackets)	Match any one character or range listed within the brackets	`[a-z]` matches every occurrence of lowercase letters `[abc]` matches every occurrence of `a`, `b`, or `c`
	`~` (tilde)	If appears immediately after the left bracket, negate the contents of the set	`[~a-z]` matches everything except lowercase letters `[a-z~A-Z]` matches all letters and the '`~`' character
Special characters	`\t` (backslash t)	Match any tab character	`\t3` matches every occurrence of a tab character followed by a `3`
	`\x` (backslash x)	Match any character specified in hex	`\x2a` matches every occurrence of the '`*`' character
	`\` (backslash)	Include the subsequent regular expression character in the search	`\-\?` matches every occurrence of ' - ' followed by '`?`'

- **Name**—Activate this option to include the variable name field of the Variables/Watch window in the search.
- **Value**—Activate this option to include the value field of the Variables/Watch window in the search.
- **Type**—Activate this option to include the variable type field of the Variables/Watch window in the search.
- **Button Bar**—Use this option to enable or disable the built-in dialog box for interactive searching. **Find Prev** and **Find Next** search for the closest previous or next occurrence of the specified text. **Stop** terminates the search, leaving the highlight on the current line. **Return** terminates the search, moving the highlight to where you initiated the search.

Edit»Replace

The **Replace** command operates the same as the **Find** command except that you can replace one search string with another string. Enter the text you want to find in the **Find What** text box and enter into the **Replace With** text box the new text you want to appear. As LabWindows/CVI performs the search, a button bar appears.

You can bypass the Replace dialog box using the keyboard commands. Refer to Appendix A, *Source Window Keyboard Commands*.

Find Next skips to the next occurrence of the search string without making a change.

Replace executes the replacement.

Replace All finds and replaces all occurrences of the specified text without asking for confirmation.

Stop terminates the search, leaving the keyboard cursor at the current position.

Return terminates the search leaving the keyboard cursor at the position where you initiated the search.

You can bypass the Replace button bar using the keyboard commands in Appendix A, *Source Window Keyboard Commands*.

Edit»Next File

If you have selected the **Multiple Files** option from either the Find or Replace button bars, you can move to the next file in the search list using this command.

View Menu

This section explains how to use the commands in a Source and Interactive Execution window **View** menu, as shown in View Menu. Use commands in the **View** menu to display line numbers and tags on source code, step through build errors, and manipulate function panels that pertain to your editing session.

View»Line Numbers

The **Line Numbers** command controls the presence of line numbers in a window. A checkmark appears next to the **Line Numbers** item in the **View** menu when you activate the line number display.

View»Line Icons

The **Line Icons** command controls the presence of line icons in a window. Line icons indicate the lines that you mark for breakpoint and the lines that you tag. A checkmark appears next to the **Line Icons** item in the **View** menu when you activate the line icons display.

Note LabWindows/CVI saves line icons in the project file. Editing source files outside of LabWindows/CVI, however, might invalidate the associated line icons.

View»Toolbar

Use the **Toolbar** command to toggle between viewing or not viewing the Function Panel window toolbar.

View»Line

The **Line** command moves the cursor to the line that you specify. When you select the **Line** command, a dialog box appears in which you enter the number of the line where you want to position the cursor.

If you specify a line number greater than the total number of lines in the program, the cursor moves to the last line of the program.

View»Beginning/End of Selection

The **Beginning/End of Selection** command toggles the window between the beginning and the end of a highlighted block of text. This is useful when you want to verify a selected block of text that is larger than the Source window.

View»Toggle Tag

The **Toggle Tag** command toggles the tag associated with the active line. Use tags to mark lines of code that you want to revisit quickly.

View»Next Tag

Use the **Next Tag** command to go to the next tagged line. Selecting **Next Tag** repeatedly takes you to all tagged lines in the windows you specify using the **Tag Scope** command.

View»Previous Tag

Use the **Previous Tag** command to go to the previous tagged line. Selecting **Previous Tag** repeatedly takes you to all tagged lines in the windows you specify using the **Tag Scope** command.

View»Tag Scope

Use the **Tag Scope** command to set which files you want to search with **Next Tag** and **Previous Tag**. You can set the scope to the current window, all open windows, or all files.

View»Clear Tags

Use the **Clear Tags** command to selectively remove existing tags.

View»Function Panel Tree

The **Function Panel Tree** command displays the Select Function Panel dialog box for the most recently used function panel.

View»Recall Function Panel

When you are editing a function call in a Source window or the Interactive Execution window, you might want to display the function panel corresponding to the call. You can do this with the **Recall Function Panel** command. The **Recall Function Panel** command not only finds and displays the panel but also sets the panel controls so that they contain the parameter values that appear in the function call. After modifying one or more controls, you can replace the original call with the modified call.

Invoking the Recall Function Panel Command

Before you invoke the **Recall Function Panel** command, you must indicate the function panel you want to recall. The simplest method is to place the cursor on a line that contains a function call or a portion of a function call. Also, you can select, or highlight, a range of lines that contain one or more function calls. You can select part of a line, provided that the part contains a function call.

If a line contains multiple function calls or one function call embedded within another, you can resolve the ambiguity by placing the cursor on or immediately after the function name.

After you indicate the function call, select the **View»Recall Function Panel**. The function panel for that function appears, and the controls contain the parameter values from the call.

Recalling a Function Panel from a Function Name Only

You can recall a panel from a function name without specifying any of the parameters. If you place the keyboard cursor on or immediately after a function name, **Recall Function Panel** recognizes the function name even if a parameter list does not follow it. Thus, you can simply type a function name into the Source window and execute **Recall Function Panel**.

Also, you can use the **Find Function Panel** command to open a function panel from a function name or a portion of a function name. Refer to the *View»Find Function Panel* section for more information.

Multiple Panels for One Function

If the selected function appears in more than one function panel window, LabWindows/CVI displays a list of panels. Select one by highlighting the panel name and pressing <Enter> or by double-clicking on the panel name.

Multiple Functions in One Function Panel Window

If the selected function matches a function panel window that contains multiple function panels, LabWindows/CVI attempts to match the panel to function calls on the lines surrounding the selected call. After the panel appears, you can check how many lines were matched to the function panel window by looking at the Source window. LabWindows/CVI highlights the matched lines.

If you select multiple lines before executing the **Recall Function Panel** command, all function calls in the selected lines must appear in one function panel window, and the order in which the window generates the calls must be identical to the order in which they appear in the selected lines. Otherwise, an error message appears.

Syntax Requirements for the Recall Function Panel Command

You do not have to compile the file you are working in before you invoke the **Recall Function Panel** command. In fact, the function call you select does not have to be syntactically valid. The only requirement is that you must spell and capitalize the name of the function correctly. If you do not spell and capitalize the function name correctly, LabWindows/CVI displays an error message indicating that the panel could not be found.

View»Find Function Panel

When you select the **Find Function Panel** command, a dialog box appears in which you can enter the name of a function. You can enter just a substring, and the **Find Function Panel** command finds all functions that contain that substring anywhere in their names. For instance, if you enter `ctrl` and click on **OK**, a dialog box appears with a list of functions including `NewCtrl`, `SetCtrlVal`, `GetCtrlVal`, and so on.

You can use a regular expression as your search string. Refer to Table 5-1 for a list of regular expression characters.

If a function panel exists for the function, LabWindows/CVI displays the panel. If two or more function panels exist for the function, LabWindows/CVI displays a list of the function panels.

The shortcut key for **Find Function Panel** is <Ctrl-Shift-P>.

View»Find UI Object

You use the **Find UI Object** command to move directly from a Source window to a User Interface Editor window. To use it, place the cursor on the constant name or callback function name of the user interface panel, control, or menu object you want to view. Then select **View»Find UI Object**. LabWindows/CVI searches each `.uir` file that is currently open or in the project for user interface objects with a matching constant name or callback function name. If LabWindows/CVI finds an object, the User Interface Editor window that contains the object comes to the foreground.

If the matching object is a panel, the panel title bar briefly flashes and the panel becomes active. If the object is a control, **Find UI Object** selects the control. If **Find UI Object** finds a menu object or more than one matching object, a dialog box that contains the list of matches appears. In this dialog box you can view information about each of the objects or select one to edit.

Build Menu

This section explains how to use the commands in a Source or Interactive Execution window **Build** menu. Use the commands in the **Build** menu to compile files and to build and link projects.

Build»Compile File

You must compile your source code before executing it in a Source window or the Interactive Execution window. The **Compile File** command adds the file to the project if necessary, checks it for syntax errors, and compiles it. If the file has any build errors after LabWindows/CVI completes compilation, a Build Errors dialog box appears.

When you want to call a function that you define in a Source window from another Source window, from the Interactive Execution window, or from a function panel, you must first execute the **Compile File** command in the Source window where you define the function. If you subsequently modify the function, you must recompile the Source window before calling the function again.

Refer to the *Options»Build Options* section in Chapter 3, *Project Window*, for more information about compiler options.

Build»Create Debuggable Executable

The **Create Debuggable Executable** menu item appears if you select the **Debug** item in the **Configuration** submenu and the **Executable** item in the **Target Type** submenu.

Use this menu item to compile and build an executable with debugging information. Use the **Debugging level** control on the Build Options dialog box to select the degree of debugging information you want LabWindows/CVI to generate for the executable. For information on the **Debugging level** control settings, refer to the *Options»Build Options* section in Chapter 3, *Project Window*. To debug the executable you create with this command, use the **Debug** menu item in the **Run** menu of a Project, Source, or Variables window.

You can specify the filename of the executable, as well as other executable settings, by selecting the **Target Settings** menu item. You can set other compile and run options using the menu items in the **Options** menu of the Project window.

Build»Create Debuggable DLL

The **Create Debuggable Dynamic Link Library** menu item appears if you select the **Debug** item in the **Configuration** submenu and the **Dynamic Link Library** item in the **Target Type** submenu.

Use this menu item to compile and build a DLL with debugging information. Use the **Debugging level** control on the Build Options dialog box to select the degree of debugging information you want LabWindows/CVI to generate for the executable.

You can specify the filename of the DLL, as well as other DLL settings, by selecting the **Target Settings** menu item. You can set other compile and run options using the menu items in the Options menu of the Project window.

The **Debug** command also generates a DLL import library for the DLL. For information on debugging DLLs, refer to the *Debugging DLLs* section in Chapter 3, *Project Window*.

Build»Create Release Executable

The **Create Release Executable** menu item appears if you select the **Release** item in the **Configuration** submenu and the **Executable** item in the **Target Type** submenu.

Use this menu item to compile and build an executable without debugging information. This command ignores the value of the **Debugging level** control on the Build Options dialog box.

You can specify the filename of the executable, as well as other executable settings, by selecting the **Target Settings** menu item. You can set other compile and run options using the menu items in the **Options** menu of the Project window.

Build»Create Release DLL

The **Create Release Dynamic Link Library** menu item appears if you select the **Release** item in the **Configuration** submenu and the **Dynamic Link Library** item in the **Target Type** submenu.

Use this menu item to compile and build a DLL without debugging information. This command ignores the value of the **Debugging level** control on the Build Options dialog box.

You can specify the filename of the DLL, as well as other DLL settings, by selecting the **Target Settings** menu item. You can set other compile and run options using the menu items in the **Options** menu of the Project window.

This command also generates a DLL import library for the DLL.

Build»Create Static Library

The **Create Static Library** menu item appears if you select the **Release** item in the **Configuration** submenu and the **Static Library** item in the **Target Type** submenu. The **Release** item is always checked.

Use this menu item to compile and build a static library without debugging information. This command ignores the value of the **Debugging level** control on the Build Options dialog box.

You can specify the filename of the static library, as well as other static library settings, by selecting the **Target Settings** menu item. You can set other compile and run options using the menu items in the **Options** menu of the Project window.

If you include a `.lib` file in a static library project, LabWindows/CVI includes all object modules from the `.lib` in the static library. This differs from creating an executable or DLL, in which LabWindows/CVI includes only the `.lib` modules that other modules in the project reference. In addition, LabWindows/CVI reports an error if you attempt to build a static library when you have a DLL import library in your project.

Build»Mark File for Compilation

When LabWindows/CVI marks a source file for compilation, a *C* appears next to the filename in the Project window. LabWindows/CVI recompiles marked files the next time you build the project. When you modify a source file, LabWindows/CVI automatically marks the file for compilation. You can force LabWindows/CVI to compile a source file on the next build with the **Mark File for Compilation** command.

Build»Clear Interactive Declarations

Variables you declare in the Interactive Execution window remain in effect until you explicitly remove them. This feature lets you use these variables in succeeding executions of the Interactive Execution window. It also enables different function panels to access the same variables.

When you delete the entire contents of the Interactive Execution window by selecting **Edit»Clear Window**, LabWindows/CVI removes the variables. If you want to remove the variables without deleting the contents of the Interactive Execution window, use the **Clear Interactive Declarations** command.

Build»Insert Include Statements

The **Insert Include Statements** command invokes a dialog box you can use to select one or more header files to include at the top of the program.

Build»Add Missing Includes

If, when you last attempted to compile the source file, the compiler reported that function prototypes were missing, **Add Missing Includes** can find include (`.h`) files that contain some or all the missing prototypes. It inserts `#include` statements for these files into your source file at the current cursor position. LabWindows/CVI adds `#include` statements only for libraries or instrument drivers that appear in the **Instrument** or **Library** menu.

Build»Generate Prototypes

After you compile a source file, you can use the **Generate Prototypes** command to generate a file that contains declarations for global and static functions and external declarations for global variables. The command generates the file into a new Source window. You can copy these declarations into your source and header files.

Build»Next/Previous Build Error

If, when you compile a file or build your project, LabWindows/CVI displays multiple errors, you can use the **Next Build Error** command to step to the next build error. LabWindows/CVI highlights source code as you step through the errors. You can use the **Previous Build Error** command to step to the previous build error.

Build»Build Errors in Next File

If, when you build your project, LabWindows/CVI displays errors for multiple files, you can use the **Build Errors in Next File** command to step to your next file with build errors. LabWindows/CVI highlights source code as you step through the errors.

Run Menu

This section explains how to use the commands in a Source or Interactive Execution window **Run** menu. Use the commands in the **Run** menu to run and debug your program.

Introduction to Breakpoints and Watch Expressions

It is important to understand the concepts of breakpoints and watch expressions before you learn about the commands in the **Run** menu.

You can pause the execution of a program without aborting it altogether by marking breakpoints in your code. You can use these breakpoints to interrupt program execution for debugging. Breakpoints can be either conditional or unconditional. Breakpoints apply to specific lines of code, but LabWindows/CVI maintains them separately from your source file. If you modify your source code outside of the LabWindows/CVI environment, you might invalidate breakpoint position information. If you set a breakpoint on a line that contains no executable code, a dialog box appears when you attempt to debug the project. The dialog box contains options to delete the breakpoint, disable the breakpoint, terminate debugging, or move the breakpoint to the next line that contains code. You can use the `Breakpoint` function to insert a breakpoint directly in your source file.

You also can use watch expressions for debugging. With watch expressions, you can specify that LabWindows/CVI suspend execution conditionally without regard to a specific line of code.

Note Breakpoints and watch expressions apply *only* to source code modules. You cannot set breakpoints in include files.

Note Some system functions on some systems might break execution when called with invalid arguments. For example, `CA_FreeMemory` might break execution when called with a pointer that was not allocated.

Breakpoint State

When a program reaches a breakpoint, LabWindows/CVI positions the keyboard cursor at the next program statement to execute and outlines the statement. You cannot edit the source code in the window while the breakpoint is in effect. However, you can use many other features of the LabWindows/CVI environment. For instance, you can look at other windows, change the state of breakpoints, and modify the value of variables in the Variables, Array Display, and String Display windows. Also, if you are at a breakpoint in a Source window, you can execute code in the Interactive Execution window or in a function panel.

To resume the execution after a breakpoint, you have several options under the **Run** menu. You can restart the project at a breakpoint by selecting **Debug**. To halt the execution of a program at a breakpoint, select **Terminate Execution** or press <Ctrl-F12> while a LabWindows/CVI environment window is active.

Setting and Clearing Breakpoints

You can set and clear breakpoints in the following ways.

- If you select **View»Line Icons**, click the mouse in the line icon area next to a line of code to set or clear a breakpoint on that line.
- Move the cursor to the line of code where you want to set or clear a breakpoint and select **Run»Toggle Breakpoint** or press <Shift-F9>.
- Select **Run»Breakpoints** to edit all breakpoints in the project and the Interactive Execution window. You also can use the **Breakpoints** command to set conditional breakpoints. Refer to the *Conditional Breakpoints* section for information about conditional breakpoints.
- Select **Run»Break at First Statement** to break on the first executable statement in the project or the Interactive Execution window.
- You can set breakpoints directly in your source code using the `Breakpoint` function.
- You can manually suspend execution while your program runs if your program checks for user input. For example, if your program makes calls to `RunUserInterface` or `scanf`, pressing <Ctrl-F12> while a LabWindows/CVI environment window is active causes a breakpoint state.

Conditional Breakpoints

Set conditional breakpoints by selecting **Run»Breakpoints**. When you assign a conditional breakpoint to a line in your program, LabWindows/CVI evaluates an expression you supply, such as `x==100` or `y<0`, before executing the line. If the expression is true, program execution suspends.

If you assigned these expressions to line 23 in your program, you would have to define `x` and `y` before line 23.

Watch Expressions

You can use watch expressions to suspend program execution conditionally. Watch expressions do not apply to specific lines of code. Instead, LabWindows/CVI evaluates them before each statement in your source code. Refer to Chapter 10, *Variables and Watch Windows*, for more information about watch expressions.

Debug/Run Interactive Statements

Running in a Source Window

If the **Target Type** is **Executable**, the **Debug** menu item runs the project's target executable for the currently selected configuration. You set the active configuration using the **Configuration** submenu in the **Build** menu. If the **Target Type** is **Dynamic Link Library**, the **Debug** menu item runs the executable you specify with the **External Process** menu item. Before running the executable, the **Debug** menu item compiles any source files that must be compiled and builds the project's target executable or DLL if you made changes since you last built the target DLL or executable.

Running in the Interactive Execution Window

Select **Run Interactive Statements** in the Interactive Execution window to execute code in that window. You do not have to enter a complete program in the Interactive Execution window. For instance, you can execute variable declarations and assignment statements in C without declaring a `main` function.

You can use the Interactive Execution window to test portions of code before including them in your main program. You also can use the Interactive Execution window to execute functions exported by a loaded instrument or by a file in the project if the project has been linked. The Interactive Execution window can access functions and data declared as global in a Source window, but a Source window cannot access the functions and data declared in the Interactive Execution window.

Refer to the *Interactive Execution Window* section for the rules governing code execution in the Interactive Execution window.

LabWindows/CVI does not disturb asynchronous I/O, RS-232 ports, opened files, and User Interface Library resources you use in the Interactive Execution window at the beginning or end of execution in the Interactive Execution window. LabWindows/CVI terminates, closes, or deletes these program elements only when one of the following events occur:

- You select **Build»Clear Interactive Declarations**.
- You select **Edit»Clear Window**.
- You link a project.
- You run a project.

Run-Time Error Reporting

LabWindows/CVI reports various run-time errors during the execution of a program. One example of a run-time error is a call to a LabWindows/CVI library function with an array or string that is too small to hold the output data.

When such errors occur, a dialog box appears identifying the type of error and the location in the file where the error occurred. LabWindows/CVI then displays the error in the Run-Time Errors window.

LabWindows/CVI suspends the program so you can inspect the values of variables in the Variables window. To terminate a program that suspended because of a run-time error, select **Run»Terminate Execution** or use the shortcut key <Ctrl-F12> while a LabWindows/CVI environment window is active.

Run»Continue

Use the **Continue** command to resume program execution when in a breakpoint state.

Run»Go To Cursor

When the program is in a breakpoint state, you can move the keyboard cursor to a line in the program and select **Run»Go To Cursor**. Program execution then continues until it reaches that line, where it enters another breakpoint state.

Run»Step Over

Use the **Step Over** command to execute an outlined statement when in a breakpoint state. If the program last suspended on a function call statement, **Step Over** executes the entire function and then enters a breakpoint state on the statement following the function call. If LabWindows/CVI encounters a breakpoint within the function call, **Step Over** pauses at the breakpoint.

Run»Step Into

The **Step Into** command is similar to the **Step Over** command except that after the program suspends operation at a function call, **Step Into** enters the function and suspends at the first statement of the function. **Step Into** can enter a function only if you define it in a source file. Otherwise, **Step Into** executes the entire function and suspends execution on the statement following the function call.

Run»Finish Function

The **Finish Function** command resumes execution through the end of the current function and breakpoints on the next statement.

Run»Terminate Execution

The **Terminate Execution** command terminates a program that is suspended at a breakpoint. The shortcut key for terminating execution of a suspended program or suspending a running program while a LabWindows/CVI environment window is active is <Ctrl-F12>.

Run»Break at First Statement

Break at First Statement is a run mode that enters a breakpoint state on the first executable statement in your source code. When activated, LabWindows/CVI puts a checkmark beside this command in the menu.

Run»Toggle Breakpoint

The **Toggle Breakpoint** command toggles the state of the breakpoint on the current line.

Run»Breakpoints

The **Breakpoints** command opens the Breakpoints dialog box, which contains a list of the breakpoints in the project. Also, you can open this dialog box by right-clicking in the line icons column and selecting **Breakpoints** from the pop-up menu.

Use the **Add/Edit Item** button to edit a single breakpoint with the Edit Breakpoint dialog box. The Edit Breakpoint dialog box contains the following items:

- **File**—Select the source file that contains the breakpoint you want to edit.
- **Line**—Select the line that contains the breakpoint you want to edit.
- **Pass Count**—Select the number of times that the source code line executes before the breakpoint occurs.
- **Condition**—Enter an optional expression that LabWindows/CVI evaluates before it executes the source code line. If the condition is true, your program enters a breakpoint state; otherwise, execution continues. Refer to the *Conditional Breakpoints* section for examples of conditional expressions.
- **Disabled** —Disable the breakpoint. The breakpoint icon in the Source window changes color to indicate that you disabled it.
- After you set all the breakpoint attributes in the Edit Breakpoint dialog box, you can **Replace** the breakpoint with the new attributes, **Add** the breakpoint to the breakpoint list, or **Cancel** the operation.

The **Go to Line** button takes you to the source code location of the currently selected breakpoint.

The **Delete Item** button deletes the currently selected breakpoint.

The **Delete All** button deletes all the breakpoints.

The **Disable All** button forces LabWindows/CVI to ignore all the breakpoints. The breakpoint icons in the Source window change color to indicate that you disabled them.

The **Enable All** button activates all the breakpoints. The breakpoint icons in the Source window change color to indicate that you enabled them.

The **OK** button accepts the current breakpoint attributes, and the **Cancel** button cancels the current operation.

Run»Stack Trace

You can use the **Stack Trace** command only when in a breakpoint state. **Stack Trace** opens a dialog box that lists the currently active functions in the program, displaying the most recently called function at the top and the initial function at the bottom. If you highlight a function in the list and select **Display**, a Source window appears with the file that contains that function. LabWindows/CVI highlights the last statement that your program executed in that function.

Run»Up Call Stack

You can use the **Up Call Stack** command only when in a breakpoint state. **Up Call Stack** moves up one level in the function call stack.

Run»Down Call Stack

You can use the **Down Call Stack** command only when in a breakpoint state. **Down Call Stack** moves down one level in the function call stack.

Run»View Variable Value

View Variable Value is a convenient way to view the contents of arrays, structures, and global variables that appear in source code. Highlight the variable that you want to see and select **View Variable Value**. Depending on the type of the variable, the Variables, Array Display, or String Display window appears with your selected variable highlighted.

Run»Add Watch Expression

Add Watch Expression is a convenient way to view the value of an expression that appears in source code. Highlight the expression that you want to see and select **Add Watch Expression**. The Watch window appears with the expression you selected.

Run»Threads

The **Threads** command brings up a dialog box listing the threads in the program you are debugging. Use this dialog box to select the threads whose local variables and call stack you want to view. When you select a thread from this dialog box and click on **OK** to close the dialog box, LabWindows/CVI displays the local variables for the selected thread in the variable display and displays the current source position of the thread in a Source window. The **Stack Trace**, **Up Call Stack**, and **Down Call Stack** commands in the Source windows **Run** menu display information on the currently selected thread. The watch display shows the thread-specific values of the expressions in the Watch window.

Instrument Menu

The **Instrument** menu is a dynamic menu. It contains a list of the loaded instrument drivers and commands to load, unload, and edit instruments. When you load an instrument, its name appears in the list. When you unload an instrument, its name disappears from the list. When you select an instrument name in the **Instrument** menu, you can access its function panels. Refer to the *Instrument Menu* section in Chapter 3, *Project Window,* for more descriptions of Instrument Menu commands.

Library Menu

This section explains how to use the commands in the **Library** menu. Use the **Library** menu commands to access function panels for the LabWindows/CVI libraries. Use library function panels to interactively run library functions and insert these function calls into any open Source window. Refer to each library overview in the *Library Reference* section of the **LabWindows/CVI Help**.

When you select a library name in the **Library** menu, you can access the library function panels. For more information, refer to the *Accessing Function Panels* section in Chapter 6, *Using Function Panels*.

Tools Menu

This section explains how to use the commands in a Source window **Tools** menu.

Tools»Create ActiveX Controller

Refer to *Create ActiveX Controller* topic in the **LabWindows/CVI Help** for more information.

Tools»Create ActiveX Server

You can use the LabWindows/CVI Create ActiveX Server Wizard to provide settings for your ActiveX Server project. Refer to the *Create ActiveX Server Wizard* topic in the **LabWindows/CVI Help** for more information. Also refer to the *Building ActiveX Servers in LabWindows/CVI* white paper from **Start»Program Files»National Instruments»Measurement Studio»Help»Measurement Studio Library**.

Tools»Edit ActiveX Server

Use the **Edit ActiveX Server** dialog box to create and modify objects and interfaces in your ActiveX server. Refer to the *Edit ActiveX Server* topic in the **LabWindows/CVI Help** for more information. Also refer to the *Building ActiveX Servers in LabWindows/CVI* white paper from **Start»Program Files»National Instruments»Measurement Studio»Help»Measurement Studio Library**.

Tools»Create IVI Instrument Driver

Select **Tools»Create IVI Instrument Driver** to open the Instrument Driver Development Wizard, to create the source file, include file, and function panel file for controlling an instrument. You can base the new instrument driver on one of the following:

- An existing driver for a similar instrument
- The core IVI driver template
- An IVI instrument class template

The Instrument Driver Development Wizard copies the template or existing driver files and replaces all instances of the original instrument prefix with the prefix you select for your new driver.

Refer to the *Measurement Studio LabWindows/CVI Instrument Driver Developers Guide* for more information on the Instrument Driver Development Wizard.

Tools»Edit Instrument Attributes

Use the **Edit Instrument Attributes** command to add, delete, or edit attributes for an IVI instrument driver. You can invoke this command only if the file in the Source window has the same path and base filename as an instrument driver function panel (`.fp`) file and its associated `.sub` file. The command is useful only if you used the **Create IVI Instrument Driver** command to generate the instrument driver files.

The **Edit Instrument Attributes** command analyzes the instrument driver files to find all the attributes the driver uses. It then opens a dialog box that displays the attributes and various information about them. In the dialog box, you can add or delete attributes, modify their properties, and enter help text for them. When you apply the changes, the command modifies the source, include, and function panel files for the instrument driver.

If you invoke the command when the text cursor is over the defined constant name or callback function name for one of the attributes, the dialog box appears with that attribute selected in the list box.

Refer to the *Measurement Studio LabWindows/CVI Instrument Driver Developers Guide* for more information about the **Edit Instrument Attributes** command.

Tools»Edit Function Tree

Use **Edit Function Tree** command to display the Function Tree Editor window for the function panel (`.fp`) file associated with the file in the Source window. The function panel file must have the same path and base filename as the file in the Source window.

Tools»Edit Function Panel

Use the **Edit Function Panel** command to display the Function Panel Editor window for a function defined in an instrument driver source file. You can use this command only if the file in the Source window has the same path and base filename as an instrument driver function panel (`.fp`) file. The text cursor must be over the name of a function for which there is a function panel in the `.fp` file.

Tools»Source Code Control

Refer to the *Tools Menu* section in Chapter 3, *Project Window,* for a description of this command.

Tools»Source Code Browser

Refer to the *Tools Menu* section in Chapter 3, *Project Window,* for a description of this command.

Window Menu

Use commands in the **Window** menu to bring any open window to the front for viewing or editing. Refer to the *Window Menu* section in Chapter 3, *Project Window*, for a description of the commands in the Window Menu.

Options Menu

Use the commands in the **Options** menu to set up preferences in the LabWindows/CVI environment and execute various LabWindows/CVI utilities.

Options»Editor Preferences

You can use the Editor Preferences dialog box to set up Source window editor preferences.

- **Undoable Actions Per File (Next Session)**—Use this option to set the number of actions per file that you can undo.
- **Purge Undo Actions When Saving File**—Use this option to clear the accumulated list of editing actions each time you save a file.
- **Move Cursor to the End of Pasted Text**—Use this option to put the cursor at the end of the pasted text. Leave this option blank to put the cursor at the beginning of the pasted text.
- **Tab Length**—Use this option to set the tab length. Activate the options to request LabWindows/CVI to convert tab characters into spaces when saving files and convert leading spaces to tab characters when loading files. These options are convenient if you use another editor or a printer that does not support tab characters.
- **Line Terminator for This File and Line Terminator for New Files**—LabWindows/CVI can read source files with any of the commonly used line-termination sequences. It remembers what line-termination sequence was found in each file and uses the same sequence when saving each file. If you want to change that sequence because you want to load the file into another editor, use the **Line Terminator** option as follows:
  - If you want to load your text file into DOS/Windows editors, select CR/LF termination.
  - If you want to load your text file into a Macintosh editor, select CR termination.

Options»Toolbar

Use the **Toolbar** command to select which icons appear in the Source window toolbar.

Options»Bracket Styles

The **Bracket Styles** command allows you to set the location of curly brackets when the following commands generate them in your program.

- The **Edit»Insert Construct** command in a Source window.
- The **Code»Generate** command in the User Interface Editor window.

You can specify two bracket styles: one for functions and another for statements, such as `if` and `switch` statements.

Options»Font

Use the **Font** command to select the font and font size for text in Source windows, Interactive Execution windows, and Variables windows. You can select from a list of monospace fonts.

Options»Colors

Use the **Colors** menu item to select colors for the Project window, Source window, Interactive Execution window, Standard I/O window, Watch window, Variables window, String Display window, and Array Display window. The **Colors** menu item does not affect dialog boxes, function panels, and the User Interface Editor window. Refer to the *Options Menu* section in Chapter 3, *Project Window,* for more information.

Options»Syntax Coloring

When you enable the **Syntax Coloring** command, LabWindows/CVI color codes the various types of tokens in your source and include files. The following are different types of tokens that can be color coded.

- C keywords
- Identifiers
- Comments
- Integers
- Real numbers
- Strings
- Preprocessor directives
- User-defined tokens

Set the color for a token type by selecting **Options»Colors**.

Create the list of user-defined tokens by selecting **Options»User Defined Tokens for Coloring**.

Options»User Defined Tokens for Coloring

Use the **User Defined Tokens for Coloring** command to define tokens for display in a unique color when you enable the **Syntax Coloring** command. Use the **Colors** command to set the color. Each token must be in the form of a valid C identifier. For each token, you can choose whether to save it in the project file or from one LabWindows/CVI session to another.

Options»Translate DOS LW Program

Use the **Translate DOS LW Program** command to convert a source file written in LabWindows for DOS so that it can run in LabWindows/CVI.

Options»Generate DLL Import Source

This command generates source code that you can use to create a DLL import library. In general, it is not necessary to use this command. For most cases, you can generate a DLL import library directly using the **Generate DLL Import Library** command. Use this command only when you must do special processing in the DLL import library. LabWindows/CVI never requires such special processing.

LabWindows/CVI enables the **Generate DLL Import Source** command only when you have an include file in the Source window. The include file must contain declarations of all the DLL functions you want to access. When you execute the command, a dialog box appears in which you enter the pathname of the DLL.

The **Generate DLL Import Source** command generates the import library source into a new Source window. You can modify the code, including making calls to functions in other source files. Create a new project that contains the source file and any other files it references. Select **Build»Target Type»Static Library** in the Project window. Execute the **Create Static Library** command.

Note You *cannot* export variables from a DLL using the import library source code this command generates. When you want to export a variable, create functions to get and set its value or create a function to return a pointer to the variable.

Note When you edit the source code this command generates, you *cannot* use the `__import` qualifier in the function declarations in the DLL include file.

Note The import source code does *not* operate in the same way as a normal DLL import library. When you link a normal DLL import library into an executable, the operating system attempts to load the DLL as soon as the program starts. The import source code operates in such a way that the DLL does not load until the user makes the first function call into it.

Options»Generate DLL Import Library

This command generates a DLL import library. LabWindows/CVI enables **Generate DLL Import Library** only when you have an include file in the Source window. The include file must contain declarations of all the functions and global variables you want to access from the DLL. When you execute the command, a dialog box appears giving you the option to generate an import library for each of the compatible external compilers rather than just for the current compatible compiler. Enter the pathname of the DLL in the file dialog box that appears.

The **Generate DLL Import Library** command generates a `.lib` file with the same base filename as the include file. If you choose to create an import library for both compilers, LabWindows/CVI creates the files in subdirectories named `msvc` and `borland`. LabWindows/CVI creates a copy of the library for the current compatible compiler in the directory of the DLL.

Options»Generate Visual Basic Include

This command generates a Visual Basic include file from the `.h` file of an instrument driver. Use this command if you are porting an instrument driver to a DLL for use in Visual Basic.

Options»Create Object File

You can use the **Create Object File** command to compile the contents of a Source window into an object file. Compiled files consume less memory and run faster than source files. They are especially useful for instrument driver programs because they load faster. Compiled files cannot be debugged, however, and they do not have run-time error checking.

Note If the source file is in the project and you do not want to debug it, use the **Compile Without Debugging** option in the Project window rather than the **Create Object File** command. You enable the **Compile Without Debugging** option by double-clicking in the `O` icon column to the right of the source filename in the Project window.

The **Create Object File** command gives you the option of creating an object file for both of the compatible external compilers rather than just for the current compatible compiler. If you choose to create an object file for both compilers, LabWindows/CVI creates the files in subdirectories named `msvc` and `borland`. LabWindows/CVI creates a copy of the object file for the current compatible compiler in the parent directory.

You can compile your file using a third-party compiler LabWindows/CVI supports. Refer to the *Measurement Studio LabWindows/CVI Programmer Reference Manual* for more information on compatible external compilers. These compiled files are smaller and execute faster than object files LabWindows/CVI creates. You can use the **Create Object File** command if you do not have access to another compiler.

Options»Preprocess Source File

Use the **Preprocess Source File** command to open a new source window that contains the preprocessed output. LabWindows/CVI replaces simple macros, expands function macros, includes header files, and resolves conditional compilation.

Help Menu

You use the commands in the **Help** menu to access information about LabWindows/CVI. Refer to the *Help Menu* section in Chapter 3, *Project Window,* for a description of the commands in the Help Menu.

6

Using Function Panels

This section describes how to use LabWindows/CVI function panels to generate code to call functions in any of the LabWindows/CVI libraries.

A function panel is an interface to the functions in the LabWindows/CVI libraries and instrument drivers. You can use function panels to help generate and test function calls within LabWindows/CVI.

A Function Panel window generates one or more function calls with function parameters you specify in the function panel. LabWindows/CVI can execute these functions immediately in the Interactive Execution window. When you execute a function panel, LabWindows/CVI copies the generated code to the Interactive Execution window and executes it. The first time you execute a function panel for an instrument driver or library, LabWindows/CVI creates and executes an `#include` statement for the header file associated with the instrument driver or library. Refer to the *Accessing Function Panels* and *Toolbars in LabWindows/CVI* sections for more information on the relationship between a function panel and the Interactive Execution window.

Instead of executing the function call, you can choose to copy the function call code to a Source window. You can later recall the function panel from the Source window by selecting **View»Recall Function Panel** in a Source window.

Normally, you use function panels to call into instrument drivers in the **Instrument** menu and libraries in the **Library** menu. Refer to the *Using Instrument Drivers* section in Chapter 3, *Project Window*, for detailed information on the relationship between instrument drivers and function panels. Also, you can use function panels to call functions in the project, as long as the functions are declared in the Interactive Execution window. Thus, you can create function panels for functions that you call frequently, even if you do not keep the functions in a separate file. Refer to the *Measurement Studio LabWindows/CVI Instrument Driver Developers Guide* for detailed information about creating function panels.

Accessing Function Panels

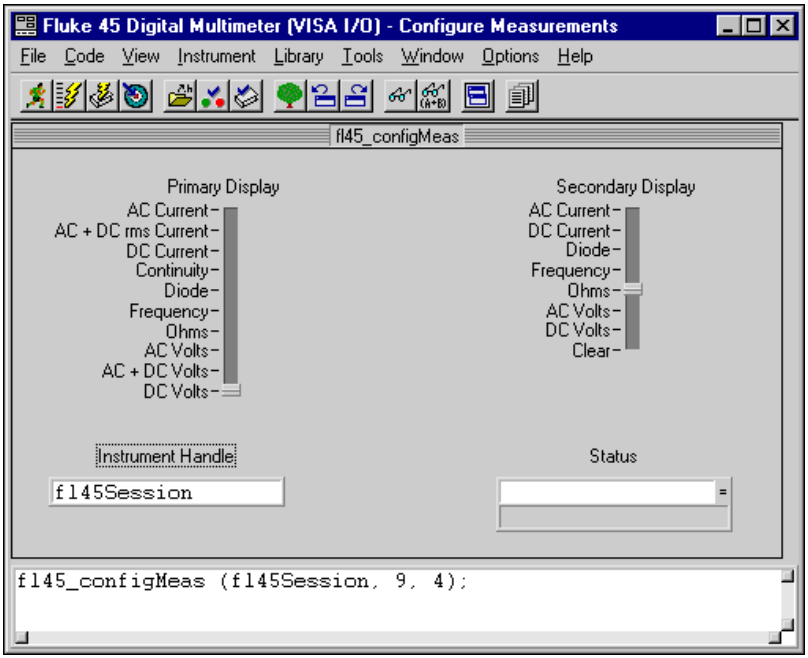
You can access a function panel for an instrument driver from the **Instrument** menu or for a library from the **Library** menu. After you select an instrument or library name, choose a panel by making selections from the Select Function Panel dialog box.

Functions are grouped in a multilevel structure called a function tree. This structure groups functions into various classes according to the operation they perform to make finding individual functions easier. When the Select Function Panel dialog box contains class names, you can select a class name to view the next level of the function tree until you reach a list of Function Panel windows.

In certain cases, it is convenient to access instrument or library module function panels in a linear fashion, that is, by moving through the list of functions without using the tree structure. The Select Function Panel dialog box has a **Flatten** option that replaces the function class hierarchy with a list of all function panels at or below the current level. Once you have selected a function panel, the function panel commands **Previous Panel**, **Next Panel**, **First Panel**, and **Last Panel** give you access to function panels in this linear manner. Refer to the *View Menu* section for more information about using these commands.

You can access function panels in other ways as well. For instance, you might want to return to a panel you recently used or recall a panel from the text of a function call in a Source window. The commands that give you access to panels in these and other ways are in the View menu of the Source window. A similar set of commands exist in the View menu of the Function Panel window.

The following figure shows the Configure Channel Function Panel window for a Hewlett-Packard 54645 Oscilloscope. It contains a function panel that corresponds to the `hp54645_ConfigureChannel` function. You can use controls on function panels to specify parameters for the functions. The generated code box at the bottom of the window displays the function calls these function panels generate.



Multiple Function Panels in a Window

The Function Panel window can contain more than one function panel. Each function panel corresponds to one function, with the controls on that function panel manipulating the parameters to that function call. You can disable individual functions by selecting **Edit»Edit Function** and selecting the **Function Disabled** checkbox from the dialog box. Disabled function calls do not appear in the generated code box, therefore, you cannot execute or insert them into a Source window.

Generated Code Box

The generated code box at the bottom of the Function Panel window displays the code the function panels produce when you manipulate the panel controls. The generated code box displays up to three lines of code at a time and is scrollable.

Toolbars in LabWindows/CVI

The LabWindows/CVI toolbar appears within function panels, in the Function Panel Editor window, and in Source windows. It gives you quick access to common commands, such as **File»Open** and **File»Save**. You can configure the toolbar to meet your needs or choose not to display it. Refer to the *Toolbars in LabWindows/CVI* section in Chapter 5, *Source and Interactive Execution Windows*, for a full description of toolbar use and configuration.

Function Panel Controls

Function panel controls specify parameters in a function call.

When you open a function panel, input from the keyboard or mouse affects the currently selected control. Pressing the <Tab> key selects the next control. Pressing <Shift-Tab> selects the previous control. To select a control with the mouse, click on the control. Pressing <Page Up> or <Page Down> moves the input focus across multiple function panels in one window. Pressing <Ctrl-Page Up> and <Ctrl-Page Down> moves from one Function panel window to the next.

The way you specify parameter values differs for each type of control. The following sections contain instructions for specifying parameters for each type of control.

Specifying a Return Value Control Parameter

A return value control displays a value that a function returns as a return value rather than as a formal parameter.

For scalar return values, you can leave the control blank. LabWindows/CVI generates a temporary variable when you run the function panel.

If you type a variable name into a return control, you must define the variable statically in the Interactive Execution window or define it elsewhere and declare it as `extern` in the Interactive Execution window before you execute the function. You can select **Code»Declare Variable** to define the variables in the Interactive Execution window. You can select **Code»Select Variable** to choose a variable or expression that you have used before. The type of value you enter must agree with the data type of the control. To determine the data type of the control, press <F1> or right-click on the control to view the Help window. After executing the function, the return value control displays the value for the variable beneath the variable name.

Specifying an Input Control Parameter

An input control accepts a value you type in from the keyboard. An input control can have a default value associated with it. This value appears in the control when the panel first appears.

To specify a parameter for an input control, select the control and type in a variable name, numeric value, or valid expression. Before executing a Function panel window, any names you type into input controls must be defined statically in the Interactive Execution window or defined elsewhere and declared as `extern` in the Interactive Execution window. You can select **Code»Declare Variable** to define variables in the Interactive Execution window for use in the function panels. You can select **Code»Select Variable** to select a variable or expression that you have used before. The type of value you enter, whether it is a constant, expression, simple variable, or array, must agree with the data type of the control. To determine the data type of the control, press <F1> or right-click on the control to view the Help window.

Specifying a Numeric Control Parameter

A numeric control behaves like an input control except that it accepts numeric values only.

If you want to type a variable name into a numeric control, select **Options»Toggle Control Style**.

Specifying a Slide Control Parameter

With a slide control you select one item from a list of options. The position of the slider, the cross-bar on the slide control, determines the value LabWindows/CVI places in the function call.

To move the slider with the keyboard, press the up or down arrow key. As you move the slider, the corresponding argument in the function call in the generated code box changes. The <Home> and <End> keys move you to the top and bottom of the slide control, respectively. To move the slider with the mouse, click on the slider and drag it up and down or just click on the position you want.

If you want to type a variable name into a slide control, select **Options»Toggle Control Style**.

Specifying a Binary Control Parameter

The *binary control* is a limited version of the slide control that has only two positions.

To select the position of the binary control, press the up or down arrow key or the <Home> or <End> key. To change the binary control with the mouse, click on the position you want.

If you want to type a variable name into a binary control, select **Options»Toggle Control Style**.

Specifying an Output Control Parameter

The *output control* displays a value that the function you execute determines.

To specify a parameter for an output control, select the control and type in the desired variable name. An output control parameter must be an array name or the address of a scalar or structure. For non-array parameters, you can leave an output control blank. LabWindows/CVI generates a temporary variable when you run the function panel. If the output control requires an array, or if you type a variable name into the output control, the variable must be defined statically in the Interactive Execution window or defined elsewhere and declared as `extern` in the Interactive Execution window before executing the function. You can use the **Declare Variable** command from the **Code** menu to define a variable in the Interactive Execution window. You can use the **Select Variable** command from the **Code** menu to select a variable or expression that you have used before.

To view the value at an output control parameter after LabWindows/CVI executes the function, double-click on the lower half of the output control to open the Variables window.

Using a Global Control

A *global control* displays the contents of global variables in a library function. You can use global controls to monitor global variables the function does not specifically return as results. These are read-only controls. You cannot alter the content, and the controls do not contribute parameters to the generated code.

Common Control Function Panel

A Function Panel window can contain a special function panel called a *Common Control* function panel. The *n* controls on a Common Control function panel specify the first ***n*** parameters of all functions in the Function Panel window.

Convenient Viewing of Function Panel Variables

Select **View Variable Value** or **Add Watch Expression** from the **Code** menu to view the contents of arrays, structures, and global variables that exist in function panel controls. Depending on the type of the variable or expression, the Variables, Array Display, String Display, or Watch window appears with the variable or expression highlighted.

File Menu

File»New, Open, Save, and Exit LabWindows/CVI

The **New**, **Open**, **Save**, and **Exit LabWindows/CVI** commands in the **File** menu of the User Interface Editor work like **New**, **Open**, **Save**, and **Exit LabWindows/CVI** commands in the Project window. For more information on these commands, refer to the *File Menu* section of Chapter 3, *Project Window*.

File»Close

The **Close** command closes the active Function Panel window.

File»Save All

The **Save All** command saves all open files to disk.

File»Add .FP File to Project

The **Add .FP File to Project** command adds the .fp file of the current Function Panel window to the project list.

File»Add Program File to Project

The **Add Program File to Project** command adds the instrument driver program file associated with the .fp file of the current Function Panel window to the project list.

File»Most Recently Closed Files

For your reference, two lists appear in the **File** menu.

- A list of the four most recently closed files, other than project files
- A list of the four most recently closed project files

Code Menu

This section contains a detailed description of the **Code** menu for Function Panel windows.

Code»Run Function Panel

Selecting the **Run Function Panel** command executes the code in the generated code box. When you select **Run Function Panel**, the following actions take place.

- LabWindows/CVI automatically inserts the header file for the library or instrument driver into the Interactive Execution window if it is not already there.
- LabWindows/CVI generates temporary variables for blank scalar output controls.
- LabWindows/CVI copies the generated function(s) to the Interactive Execution window.
- LabWindows/CVI executes the code. While executing, the **<<Running>>** menu appears in the upper left corner of the function panel menu bar.
- LabWindows/CVI displays the new values for output, return values, and global variable controls.

Code»Declare Variable

Use **Declare Variable** to declare a variable to be placed in the currently active control on the function panel.

- **Variable Type**—Indicates the data type associated with the currently active control on the panel. You can use more than one data type for some controls. In such cases, a ring control allows you to select the data type.
- **Variable Name**—Enter the name of the variable you want to declare. LabWindows/CVI automatically prefixes scalar output variables with an ampersand (`&`).
- **Number of Elements**—Appears when the currently active control is for an array or a string. Enter the number of elements.
- **Execute Declaration**—Enabling this option causes LabWindows/CVI to execute the variable declaration immediately in the Interactive Execution window.
- **Add Declaration to Top of Target File (*`filename`*)**—Enabling this option causes LabWindows/CVI to insert a copy of the declaration at the top of the file you select using the **Set Target File** button.
- **Add Declaration to Current Block in Target File (*`filename`*)**—Enabling this option causes LabWindows/CVI to insert a copy of the declaration at the beginning of the code block that contains your current position.
- **Set Target File**—Sets the destination file for the **Insert Function Call** command. The Set Target File dialog box opens a dialog box from which you can select any open Source window or the Interactive Execution window as the destination file.

- **OK**—Declares the variable according to the options you selected.
- **Cancel**—Cancels the operation and closes the Declare Variable dialog box.

When you use the **Declare Variable** command, LabWindows/CVI always declares the variable using the static storage class, unless you enable the **Add Declaration to Current Block in Target File** option.

In addition to generating the variable declaration, the **Declare Variable** command also places the variable name in the currently active control, overwriting the previous contents of the control.

If the currently active control already contains a syntactically correct variable name, it appears in the **Variable Name** text box when the Declare Variable dialog box first appears.

Code»Clear Interactive Declarations

Variables declared in the Interactive Execution window remain in effect until you explicitly remove them. Because of this feature, you can use these same variables in succeeding executions of the Interactive Execution window, and different function panels can access the same variables.

The **Clear Interactive Declarations** command removes the variables without deleting the contents of the Interactive Execution window.

Code»Select UIR Constant

The **Select UIR Constant** command can help you use the function panels for the User Interface Library. The command lets you select from the list of constant names associated with the objects in your `.uir` files.

When you specify a parameter for an input control that can accept a panel resource ID, control ID, menu bar resource ID, menu ID, or menu item ID, use **Select UIR Constant** to open the Select UIR Constant dialog box.

The list box at the top of the dialog box lists all the `.uir` files open or in the project. Only constants from the currently selected `.uir` file appear in the list box at the bottom. Click a file to select it.

- **Constant Type**—Select which category of constant name to display in the list box below **Constant Type**.
- **OK**—Copies the currently selected constant name into the function panel control.
- **Cancel**—Cancels the operation and closes the Select UIR Constant dialog box.

Note If you attempt to use **Select UIR Constant** on the Panel Handle and Menu Bar Handle controls that appear on most User Interface Library function panels, an error message appears. These controls take the values returned from `LoadPanel` and `LoadMenuBar`, so an attempt to select `.uir` constants will fail.

You can use **Select UIR Constant** in user-defined panels. That way, the command is available to function panels for user libraries that you build on top of the User Interface Library.

Code»Select Attribute Constant

In certain cases, the **Select Attribute Constant** command replaces the **Select UIR Constant** command in the **Code** menu. This occurs in panels for functions that set or get attribute values. The User Interface Library, the VISA Library, and IVI instrument drivers have such functions. Examples are `GetCtrlAttribute`, `SetCtrlAttribute`, `GetPanelAttribute`, and `SetPanelAttribute` in the User Interface Library. The panels for these functions each contain an Attribute ring control and a corresponding Value input control. When either of these two controls are active, the **Select Attribute Constant** command appears in the **Code** menu. The action of the command differs based on whether the Attribute or Value control is active.

Select Attribute Constant

When you execute the Select Attribute Constant command on an Attribute ring control, the Select Attribute Constant dialog box appears.

- **Control type**—Appears only when you access this dialog box from a User Interface Library function panel. Use this ring to restrict the list of attributes to those applicable to a particular control type.
- **Data type**—Appears when you access this dialog box from a function panel for a typesafe attribute function. The IVI Library and IVI instrument drivers have typesafe attribute functions, such as `Ivi_SetAttributeViInt32`. Use the Data type ring to restrict the list of attributes to those that have the same data type as the typesafe function. If you choose to see all attributes, the data type of each attribute appears on the right-hand side of the list box. The data type is dim if it is not the same as the data type of the typesafe function.
- **Attributes**—A list box that displays the attributes you can use with the function. The attributes are organized under classes. A trailing ellipsis (...) denotes a class. The name of the attribute is dim if it does not allow the type of access that the function performs. For example, read-only attributes for user interface controls appear dim when you access this dialog box from the function panel for `SetCtrlAttribute`.
- **Attribute Help**—Displays help information for the currently selected attribute.
- **OK**—Click OK or double click an attribute to change the function panel ring control to that attribute. If you select an attribute that appears dim in the list box, an error message

appears informing you that the attribute does not allow the type of access the function requires. If you select an attribute for which the data type appears dim in the list box, a dialog box appears giving you the option to change to the function panel for the typesafe function that you can use with that attribute.

- **Cancel**—Cancels the operation and closes the Select Attribute Constant dialog box.
- **Flatten**—Enable to see a list of all the attributes without the classes and in alphabetical order in the Attributes list box.

Notice that when you attempt to operate the Attribute ring control in the function panel as a normal ring control, the same dialog box appears in place of the pop-up menu that normally appears on a ring control.

Select Attribute Value

The **Select UIR Constant** command has special behavior on the Attribute Value input and output controls in panels for functions such as `GetCtrlAttribute`, `SetCtrlAttribute`, `GetPanelAttribute`, and `SetPanelAttribute`.

When you execute the **Select Attribute Constant** command on an Attribute Value control, the behavior depends on the attribute currently selected in the Attribute ring control on the same function panel.

If you set the Attribute ring control to an attribute for which there is no small, discrete set of values, the Attribute Value Information dialog box appears, repeating the help information for the attribute. If, on the other hand, a small, discrete set of values exists, the Select Attribute Value dialog box appears.

- **Attribute**—The attribute currently selected in the Attribute ring control on the function panel.
- **Values**—Displays the attribute values. When the values that appear in the Values list box are constant names, the actual values appear on the right-hand side of the list box.

  If the Attribute Value control is an input control, such as on `SetCtrlAttribute` or `SetPanelAttribute`, double-click on an entry in the Values list to copy it into the Attribute Value control on the function panel.

  If the Attribute Value control is an output control, such as on `GetCtrlAttribute` or `GetPanelAttribute`, and a value appears in the bottom half of the control because you executed the function panel, LabWindows/CVI selects, when possible, the value in the Values list that corresponds to the value shown in the bottom half of the output control. An arrow symbol appears to the left of the list box entry that contains that value.
- **Value Help**—Displays help for the value selected in the Values list box.
- **Attribute Help**—Displays help for the attribute currently selected in the Attribute ring control on the function panel.

Code»Select Variable or Expression

The **Select Variable** command gives you a list of previously used variables or expressions that have data types that are compatible with the currently active function panel control. LabWindows/CVI enables the command only when the currently active function panel control is one that accepts text entry. When you select a variable or expression from the list, LabWindows/CVI copies it into the function panel control. The **Select Variable** command can significantly reduce the amount of keyboard entry necessary when using function panels.

When you execute the **Select Variable** command, the Select Variable or Expression dialog box appears.

- **Data Type of Control**—Indicates the data type of the currently active function panel control.
- **Variable or Expression**—Contains the variables and expressions that have data types compatible with the data type of the control.
- **Data Type**—Indicates the data type of each variable and expression.
- **Show Project Variables**—Adds to the list box global variables, static and non-static, defined in project files that have been successfully compiled.
- **OK**—Dismisses the dialog box and copies the variable or expression into the function panel control. It might add a leading ampersand (`&`) when the function panel control is an output control. It might add one or more leading asterisks (`*`) or a trailing array indexation (`[0]`) when necessary to correctly match the data type of the control.
- **Cancel**—Cancels the operation and closes the Select Variable or Expression dialog box.

What is Included in a List Box

Select Variable considers the following items for inclusion in the list box:

- Variables you declare in the Interactive Execution window
- Variables you declare using the **Declare Variable** command in a function panel
- Variables or expressions used in function panels you execute
- Variables or expressions used in function panels from which you insert code into a Source window
- User Interface panel handle variables that CodeBuilder adds to a Source window
- Variables declared as global or static global in a project file that has been successfully compiled but only if the **Show Project Variables** option is enabled in the dialog box

LabWindows/CVI removes some or all these items from memory when you unload the current project or when you select **Build»Clear Interactive Declarations**.

Data Type Compatibility

Compatibility between data types is a more complex issue than you might expect. LabWindows/CVI uses a number of heuristics. The heuristics differ based on whether the variable is known to the compiler.

Variables known to the compiler include variables you declare in the Interactive Execution window and variables you declare in project files that you have successfully compiled. For such variables, LabWindows/CVI uses the following factors to determine whether the variable is type-compatible with a function panel control.

- LabWindows/CVI reduces data types you declare with the `typedef` keyword to their most intrinsic type, as long as the `typedef` is known to the compiler. For example, assume the compiler has processed the following declarations.

  ```
  typedef int typeA;
  typedef int typeB;
  typedef typeB typeC;
  ```

  A variable of type `typeA` is an exact match for a function panel control that has type `typeC`.

- LabWindows/CVI considers all numeric types compatible with each other except that floating-point variables or expressions are not considered compatible with integer function panel controls.

- LabWindows/CVI considers types that have the same base type but differ in levels of indirection to be compatible. For example, the following are all compatible:

  ```
  int
  int *
  int **
  int []
  ```

  To be included in the list box, an expression or a variable name the compiler does not know must match exactly the data type of the function panel control. An example of a variable name not known to the compiler is one used in a function panel from which you insert code into a Source window.

Note An expression or variable name the compiler does not know can be associated with multiple data types. For instance, you might use the same variable name in an `int` control and a `double` control. If the variable is not known to the compiler, LabWindows/CVI has no way of knowing the true data type of the variable name. Thus, you might see the variable name associated with different data types.

Sorting List Box Entries

LabWindows/CVI first sorts the entries in the list box by data type. The most compatible data types appear first. The exception is that some function panel controls use meta data types, such as `numeric array`, `any array`, or `any type`. Such controls are equally compatible with a wide range of data types. In this case, the order of data types does not indicate differing degrees of compatibility.

Within each data type, LabWindows/CVI sorts the entries alphabetically by the variable/expression text.

Code»Insert Function Call

The **Insert Function Call** command copies the generated code to the selected window at the current location of the keyboard cursor. You can copy code to any open Source window or to the Interactive Execution window. You determine the destination window by selecting **Code»Set Target File**.

If the destination window contains selected text, LabWindows/CVI displays a dialog box that gives you the option of replacing the selected text or inserting the generated code after the selected text. Refer to *View»Recall Function Panel* section in Chapter 5, *Source and Interactive Execution Windows*, for more information.

Code»Set Target File

Use the **Set Target File** command to set the destination file for the **Insert Function Call** command. **Set Target File** brings up a dialog box from which you can select from any open Source window or the Interactive Execution window.

Code»View Variable Value

Use the **View Variable Value** command to view the contents of arrays, structures, and global variables that appear in a function panel. Highlight the variable that you want to see and select **View Variable Value**. Depending on the type of the variable, the Variables, Array Display, or String Display window appears with the variable highlighted.

Code»Add Watch Expression

Use the **Add Watch Expression** command to view the value of an expression that appears in a function panel. Highlight the expression you want to see and select **Add Watch Expression**. The Watch window appears with the expression highlighted.

View Menu

This section contains a detailed description of the **View** menu for Function Panel windows.

View»Toolbar

Use the **Toolbar** command to toggle between viewing or not viewing the Function Panel window toolbar.

View»Error

If an error occurs during the execution of a function panel, you can use the **Error** command to toggle between the error message and the code in the generated code box.

View»Include File

The **Include File** command displays the include file associated with the library or instrument driver in a Source window. The include file contains all the function prototypes for the library or instrument driver.

View»Current Tree

The **Current Tree** command displays the Select Function Panel dialog box for the most recently used function panel, making it easy for you to return to the location of the current panel in the function tree.

View»Function Panel History

The **Function Panel History** command displays a scrollable list of the function panels you have used during the current LabWindows/CVI session. You can display function panels from the list as new windows, or you can overwrite the current Function Panel window.

View»Find Function Panel

When you select the **Find Function Panel** command, a dialog box appears in which you can enter the name of a function. You can enter just a substring, and the **Find Function Panel** command finds all functions that contain that substring anywhere in their names. For instance, if you enter `ctrl` and click on **OK**, a dialog box appears with a list of functions including `NewCtrl`, `SetCtrlVal`, `GetCtrlVal`, and so on.

You can use a regular expression as your search string. Refer to Table 5-1, , in Chapter 5, *Source and Interactive Execution Windows*, for a list of regular expression characters.

If a function panel exists for the function, LabWindows/CVI displays the panel. If two or more function panels exist for the function, LabWindows/CVI displays a list of the function panels.

The shortcut key for **Find Function Panel** is <Ctrl-Shift-P>.

View»Previous Function Panel

The **Previous Function Panel** command displays the previous function panel in the current Function Panel window.

View»Next Function Panel

The **Next Function Panel** command displays the next function panel in the current Function Panel window.

View»Previous Function Panel Window

The **Previous Function Panel Window** command opens the Function Panel window that precedes the current Function Panel window in the same Function Tree.

View»Next Function Panel Window

The **Next Function Panel Window** command opens the Function Panel window that follows the current Function Panel window in the same Function Tree.

The rotation order for the Function Tree is circular. If the first Function Panel window in the tree is visible on the screen, selecting **Previous Function Panel Window** displays the last Function Panel window in the tree. If the last Function Panel window in the tree is visible, selecting **Next Function Panel Window** displays the first Function Panel window in the tree.

View»First Function Panel Window

The **First Function Panel Window** command displays the first Function Panel window in the Function Tree.

View»Last Function Panel Window

The **Last Function Panel Window** command displays the last Function Panel window in the Function Tree.

Instrument Menu

The **Instrument** menu is a dynamic menu. It contains a list of the loaded instrument drivers and commands to load, unload, and edit instruments. When you load an instrument, its name appears in the list. When you unload an instrument, its name disappears from the list. When you select an instrument name in the **Instrument** menu, you can access its function panels. Refer to the *Instrument Menu* section in Chapter 3, *Project Window,* for more descriptions of Instrument menu commands.

Library Menu

This section explains how to use the commands in the **Library** menu. Use the **Library** menu commands to access function panels for the LabWindows/CVI libraries. Use library function panels to interactively run library functions and insert these function calls into any open Source window. Refer to each library overview in the *Library Reference* section of the **LabWindows/CVI Help**.

When you select a library name in the **Library** menu, you can access the library function panels. For more information, refer to the *Accessing Function Panels* section.

Tools Menu

Refer to the *Tools Menu* section in Chapter 3, *Project Window*, for information about the **Tools** menu.

Window Menu

Use commands in the **Window** menu to bring any open window to the front for viewing or editing. Refer to the *Window Menu* section in Chapter 3, *Project Window,* for a description of the commands in the Window Menu.

Options Menu

This section contains a detailed description of the **Options** menu for Function Panel windows.

Options»Default Control

Default Control resets a control to its default value and configuration.

Options»Default All

Default All resets all the controls on the current Function Panel window to their default values and configurations.

Options»Toolbar

Use the **Toolbar** command to select which icons appear in the Function Panel window toolbar. Refer to the *Toolbars in LabWindows/CVI* section of Chapter 5, *Source and Interactive Execution Windows*, for more details on editing the toolbar.

Options»Exclude Function

The **Exclude Function** command disables the current function panel so that the function call does not appear in the generated code box and is not in effect when you select **Code»Run** or **Insert**.

Options»Toggle Control Style

Slide, binary, and ring controls insert a number into a function call in the generated code box. The value of this number depends on the item you select in the control. You can override the configured values of these controls by using the **Toggle Control Style** command.

Toggle Control Style replaces a slide, binary, or ring control with an input control. You can use this input control to enter a variable name, constant, or expression. This entry appears in the generated code box in the same position as the parameter that the original control produced.

The variable name or constant that you enter must match the type specified for the control, such as short, long, single-precision, double-precision, string, and so on. Otherwise, a syntax error occurs when you execute the function.

Options»Change Format

Change Format lets you change the numeric format for scalar controls. The list of formats depends on the data type associated with the control.

You can display short and long data types in decimal, hexadecimal, octal, or ASCII form. You can display real numbers in floating-point or scientific format.

Options»Edit Function Panel Window

The **Edit Function Panel Window** command puts the Function Panel window in edit mode. The function panel window is a collection of panels that represent all functions that users can interactively call from that window. Refer to Chapter 8, *Function Panel Editor*, for information about editing instrument function panels.

Note You *cannot* edit the function panels of the LabWindows/CVI libraries or user libraries.

Help Menu

The **Help** menu for Function Panel windows works the same way as the **Help** menu in the Project window except that it also includes **Control** and **Function** help. Refer to the *Help Menu* section in Chapter 3, *Project Window,* for more information.

Control

The **Control** command displays help information about the currently highlighted control. To select control help with the mouse, right-click anywhere on the control you want help with.

Function

The **Function** command displays general information about the function the current Function Panel generates. To select function help with the mouse, right-click on the Function Panel you want help with.

7

Function Tree Editor

About the Function Tree and Function Tree Editor

The function tree defines the way functions are grouped in the dialog boxes. Users access the function panels of an instrument driver through the Select Function Panel dialog box that they select from the **Instrument** menu. Use the Function Tree Editor to create and modify the function tree for an instrument driver.

To invoke the Function Tree Editor, select **File»New»Function Tree (*.fp)** or **File»Open»Function Tree (*.fp)**.

When you invoke the Function Tree Editor, a new Function Tree Editor window appears. If you selected **New** to create a new function tree, you see a blank Function Tree Editor window. If you selected **Open** to edit an existing function tree, the function tree for the file you selected appears in the window. To edit the function panel of an instrument driver that is loaded in the **Instrument** menu, select **Instrument»Edit**. Highlight the name of the instrument in the selection list of the Edit Instrument dialog box and click the **Edit Function Tree** button.

The function tree context menu appears when you right-click a function panel window node in the Function Tree Editor.

Function Tree Editor Context Menu

The following menu options are the same or similar to options available through the menu bar selections.

- **Edit Node**—Performs the same function as the **Edit»Edit Node** command.
- **Edit Function Panel Window**—Performs the same function as the **Edit»Edit Function Panel Window** command.
- **Generate Source For Function Node**—Generates a function definition and declaration for the currently selected function node. If a function definition already exists in the driver source file, you are prompted for permission to update it. You can replace, insert above or below, or skip without updating. Your choice is then also applied to the declaration in the driver include file.

- **Go To Definition**—Opens the driver source file and jumps to the definition of the function that is currently selected in the function tree.
- **Go To Declaration**—Opens the driver include file and jumps to the declaration of the function that is currently selected in the function tree.

File Menu

Use the commands in the **File** menu to create a new function tree, edit an existing function tree, save function panel information into a `.fp` and `.sub` file on disk, or add function panels to a project.

For each IVI instrument driver, a `.sub` file accompanies the `.fp` file. The `.sub` file contains the information about the instrument driver attributes. You edit this information using the attribute editor. When you save the contents of a `.fp` file, LabWindows/CVI also saves the contents of the `.sub` file automatically.

File»New, Open, Save, and Exit LabWindows/CVI

The **New**, **Open**, **Save**, and **Exit LabWindows/CVI** commands in the **File** menu of the User Interface Editor work like **New**, **Open**, **Save**, and **Exit LabWindows/CVI** commands in the Project window. For more information on these commands, refer to the *File Menu* section of Chapter 3, *Project Window*.

File»Save .FP File

The **Save** command writes the contents of the active window to disk. If you want to append a different extension, type it in after the filename. If you do not want to append an extension, enter a period after the filename.

File»Save .FP File As

The **Save As** command writes the contents of the active window to disk using a new filename you specify and changes the name of the active window to the name you specified.

File»Save Copy of .FP File As

Use the **Save Copy of .FP File As** command to save a copy of your `.fp` using a name you specify without changing the name of the active window.

File»Close

The **Close** command closes the active window. If you have modified the contents of the window since the last save, LabWindows/CVI prompts you to save the file to disk.

File»Save All

The **Save All** command saves all open files to disk.

File»Add .FP File To Project

The **Add .FP File To Project** command adds the `.fp` file to the project list.

File»Add Program File To Project

The **Add File To Project** command adds the file in the current window to the project list.

File»Read Only

The **Read Only** command suppresses the text editing capabilities in the current window. When you initially open a file, LabWindows/CVI disables the **Read Only** command unless the file is read only on disk.

Edit Menu

Use the commands in the **Edit** menu to edit the entries in the function tree.

Edit»Cut

Cut deletes the selected function or class from the tree and copies it to the clipboard.

Edit»Copy

Copy copies the selected function or class from the tree to the clipboard.

Edit»Paste Above

Paste Above inserts the contents of the clipboard into the tree above the selected node.

Edit»Paste Below

Paste Below inserts the contents of the clipboard into the tree below the selected node.

Edit»Edit Node

Edit Node lets you edit the instrument, function, or class name on the highlighted line.

Edit»Edit Help

Edit Help lets you add context-sensitive help information to the function tree.

Edit»Edit Function Panel Window

Edit»Function Panel Window lets you edit the selected function panel window in the Function Panel Editor. The function panel window is a collection of panels that represent all functions that users can interactively call from that window.

Edit».FP Auto-Load List

.FP Auto-Load List allows you to specify other instrument drivers that the current instrument driver depends on. LabWindows/CVI loads these instrument drivers automatically when you load the current instrument driver.

The **.FP Auto-Load List** command opens a dialog box in which you can list simple `.fp` filenames. Do not include drive or directory names. When you load the current instrument driver, LabWindows/CVI tries also to load the instrument drivers identified by these `.fp` filenames.

LabWindows/CVI looks for these `.fp` files in the following sequence:

1. If the `.fp` file is under the VXI*plug&play* framework directory, LabWindows/CVI looks for the `.fp` file using the following pathnames, where *`vppfrmwk`* is the VXI*plug&play* framework directory and prefix is the instrument prefix:

   `vppfrmwk``\support\`*`prefix`*`\`*`prefix`*`.fp`

   `vppfrmwk``\`*`prefix`*`\`*`prefix`*`.fp`
2. LabWindows/CVI looks in the directory of the referencing `.fp` file.
3. LabWindows/CVI then looks for the `.fp` files in the instrument directories list. Edit the instrument directories list by selecting **Options»Instrument Directories** in the Project window.
4. Finally, LabWindows/CVI looks for the files in the `instr` directory under the directory where LabWindows/CVI is installed.

If LabWindows/CVI cannot find the `.fp` file, you can look for the file using a file dialog box. If you find the `.fp` file, LabWindows/CVI prompts you to add the directory to the instrument directories list and prompts you to add the file to the project.

If an auto-loaded `.fp` file has no classes or function panels, it does not appear in the **Instrument** menu. This is useful for support modules that contain no user-callable functions.

When you select **Instrument»Unload**, all auto-loaded `.fp` files appear in the Unload Instrument dialog box. Auto-loaded instruments are not unloaded automatically when the dependent instrument is unloaded.

Edit»Find

The **Find** command allows you to locate a particular text string in the function panel file. You can search for text in node names; function names; control labels; control values; item labels in ring, slide, and binary controls; message control text; and help text. When you search in help text, you cannot search in any of the other items at the same time. The search begins at the function tree node for the current Function Panel Editor window. The **Find** command always searches all controls on the panel regardless of whether any are currently selected.

If the **Find** command brings up a Help Editor window and you do not use the button bar, you must return to the Function Panel Editor window or Function Tree Editor window to continue searching throughout the function panel file. The **Find** command in the Help Editor window searches only within the window. You can return to the Function Panel Editor window by pressing <F8>. You can return to the Function Tree Editor window by pressing <F7>.

Edit»Replace

The **Replace** command operates the same as the **Find** command except that you can replace the search string with another search string.

Note When you cut or copy a class to the Function Tree Editor clipboard, all its subclasses and functions are cut or copied as well. Similarly, when you paste the class, all its subclasses and functions are pasted.

Create Menu

Use the commands in the **Create** menu to create a new function tree or add new functions and classes to an existing function tree.

Create»Instrument

The **Create»Instrument** command lets you create a new function tree. When you select **Instrument**, a dialog box appears. Enter the following information in the Create Instrument Node dialog box:

- The name of the instrument (up to 40 characters).
- The prefix that you want LabWindows/CVI to add to the beginning of each function name. The prefix cannot exceed eight characters. Do not include the underscore (_) separator in your prefix. LabWindows/CVI adds an underscore (_) separator to the prefix before appending the function name to it.

The instrument name you enter in the Create Instrument Node dialog box appears at the bottom of the Function Tree Editor window. The `Create Class or Function Panel Window` line appears beneath the instrument name. Add functions and classes to the function tree using the Function and Class commands.

Create»Class

Use the **Class** command to add a new class to a function tree.

When you select the **Class** command, a dialog box appears. Enter the name that you want to appear in the Select Function Panel dialog box that appears when the user selects the instrument from the **Instrument** menu.

Adding a Class to an Empty Tree or Class

Complete the following steps to add a class to an empty tree.

1. Select the line that contains `Create Class or Function Panel Window`.
2. Select the **Create»Class**. The Create Class Node dialog box appears.
3. Complete the Create Class Node dialog box. The class appears in the function tree window. The new class name takes the place of the `Create Class or Function Panel Window` message on the selected line.

Inserting a Class into an Existing Tree

In the function panel hierarchy, you can insert up to eight levels of classes. Complete the following steps to insert a class into a function tree.

1. Select an existing function or class at the level you want to place the new class.
2. Select **Create»Class**. The Create Class Node dialog box appears.
3. Complete the Create Class Node dialog box. The new class is inserted on the line below the existing function or class. The class exists at the same level in the tree as the function or class that originally occupied the line.

Note A function tree can contain a combination of up to 32,000 functions and classes.

Create»Function Panel Window

Select **Create»Function Panel Window** to add a new function to a function tree. The function panel window is a collection of panels that represent all functions that users can interactively call from that window.

When you select the **Function Panel Window** command, a dialog box appears. Enter the following information in the Create Function Panel Window Node dialog box.

1. In the Name text box, enter the name that you want to appear in the Function Panel Selection dialog box when the instrument is chosen from the **Instrument** menu.
2. In the Function Name text box, enter the actual code name used in the instrument driver for the function being added. This function name must be valid for the current language.

Note The name of every function in an instrument driver begins with a common prefix. Do not enter the prefix of the function name. LabWindows/CVI automatically adds the prefix to each function name. You specify the prefix by selecting **Create»Instrument**.

Adding a Function to an Empty Tree or Class

Complete the following steps to add a function to an empty tree or class.

1. Select the line that contains `Create Class or Function Panel Window`.
2. Select **Create»Function Panel Window**. The Create Function Panel Window Node dialog box appears.
3. Complete the Create Function Panel Window Node dialog box. The new function name appears in place of the `Create Class or Function Panel Window` message on the selected line.

Inserting a Function into an Existing Tree

Complete the following steps to insert a function at any level in an existing function tree.

1. Select an existing function or class at the level you want to place the new function.
2. Select **Create»Function Panel Window**. The Create Function Panel Window Node dialog box appears.
3. Complete the Create Function Panel Window Node dialog box. The new function is inserted on the line below the existing function or class. The function exists at the same level in the tree as the function or class that originally occupied the line.

Instrument Menu

Use the commands in the **Instrument** menu to load and edit an instrument driver and to edit a function in the loaded instrument driver. The **Instrument** menu lists the loaded instrument drivers. The instrument function tree you select appears in a Function Tree Editor window.

Instrument»Load

Select **Instrument»Load** to add a new instrument driver to the **Instrument** menu. The **Load** command operates like the **File»Open** command. When you select the **Load** command, the Load Instrument dialog box appears. Enter the appropriate information to select an existing function panel file.

Instrument»Unload

Select **Instrument»Unload** to remove one or all instrument drivers from the **Instrument** menu. When you select the **Unload** command, the Unload Instrument dialog box appears. In this dialog box, you have the following options.

- Use the mouse or the cursor keys and space bar to individually select which instrument drivers to unload.
- Select all instrument drivers by clicking the **Check All** button.
- Deselect all instrument drivers by clicking the **Check None** button.
- Click the **OK** button to unload the selected instrument drivers.
- Click the **Cancel** button to return without unloading any instrument drivers.

Instrument»Edit

The **Edit** command lets you invoke the Function Panel Editor or modify the relationship between the function panel file and its associated program file. When you select **Instrument»Edit**, the Edit Instrument Dialog Box appears.

The Edit Instrument dialog box presents the following options:

- **Show Info**—Displays the names of the current function panel file and the attached program file. It also shows whether these files are in the current project and if the program file is compiled. The attached program file contains the functions that are called when users operate the function panel.
- **Attach and Edit Source**—Searches the directory that contains the function panel file for a filename that has the same prefix as the function panel file and a `.c` extension. If the file is found, a new Source window opens with the file displayed in it, and the source file is attached to the function panel. If the file is not found, you are prompted to create a new source file and a blank Source window appears.
- **Detach Program**—Detaches the program file from the function panel.
- **Reattach Program**—Attaches a program file to a function panel. It searches the directory that contains the function panel file for a filename that has the same prefix as the function panel file and a `.obj`, `.dll`, or `.c` extension. If a file is found, the program attaches it to the function panel.
- **Edit Function Tree**—Invokes the Function Tree Editor.
- **Done**—Exits the Edit Instrument dialog box without modifying the function panel.

Tools Menu

Use the commands in the **Tools** menu to generate function definitions and declarations into your source and include files, jump to function definitions and declarations, generate `.hpp` files for the function tree, and invoke the instrument driver developer wizard and attribute editor.

Tools»Create ActiveX Controller

Refer to the *Tools Menu* section of Chapter 3, *Project Window,* for a description of this command.

Tools»Create ActiveX Server

Refer to the *Tools Menu* section of Chapter 3, *Project Window,* for a description of this command.

Tools»Edit ActiveX Server

Refer to the *Tools Menu* section of Chapter 3, *Project Window,* for a description of this command.

Tools»Create IVI Instrument Driver

This command initiates the instrument driver developer wizard. Refer to Chapter 3, *Programming Guidelines for Instrument Drivers*, in the *Measurement Studio LabWindows/CVI Instrument Driver Developers Guide,* for more information on the instrument driver developer wizard.

Tools»Edit Instrument Attributes

Edit Instrument Attributes initiates the attribute editor. Refer to Chapter 4, *Attribute Editor*, in the *Measurement Studio LabWindows/CVI Instrument Driver Developers Guide,* for more information on the attribute editor.

Tools»Generate IVI C++ Wrapper

Generate IVI C++ Wrapper generates a `.hpp` file for the function tree. This file contains the definition of a C++ class for the instrument driver. A wrapper function is generated for each function in the function tree. Required C++ member functions, such as constructors and destructors, are also generated. The generated class references classes that are defined in `ivibase.hpp` and `iviexcpt.hpp`, which are located in the `cvi\include` directory. You can edit these classes to customize the generated wrapper class. This command is available only for IVI drivers.

Tools»Enable Auto Replace

Enable Auto Replace enables the updating of instrument driver source files to reflect changes to function names in the function tree. This option is global to LabWindows/CVI, so enabling it in one Function Tree Editor window enables it for all Function Tree Editor windows. This command is dimmed for function trees that have not yet been saved.

When this option is enabled and not dimmed, LabWindows/CVI updates the instrument driver `.c`, `.h`, and `.sub` files to reflect changes you make to function names or the instrument prefix in the Function Tree Editor window or Function Panel Editor window. When you change a function name, LabWindows/CVI prompts you for permission to update your instrument driver to reflect the new name. When you change the instrument prefix, LabWindows/CVI prompts you for permission to update your instrument driver to reflect the new prefix.

Tools»Generate New Source For Function Tree

Generate New Source For Function Tree generates function definitions and declarations into your driver source and include files. For each function panel in the function tree, the source (`.c`) file is searched for the function definition. The header (`.h`) file is searched for the function declaration. If a function definition or declaration cannot be found in the appropriate file, an empty function shell is generated in the file.

Tools»Go To Definition

Go To Definition opens the driver source file and jumps to the definition of the function that is currently selected in the function tree.

Tools»Go To Declaration

Go To Declaration opens the driver include file and jumps to the declaration of the function that is currently selected in the function tree.

Tools»Source Code Control

Source Code Control contains menu items that you can use to perform operations with your source code control system. LabWindows/CVI does not provide a source code control system. If you have one that implements the standard Source Code Control Interface, you can attach a LabWindows/CVI project to your source code control system using **Options»Source Code Control Options** in the Project window.

Window Menu

Use commands in the **Window** menu to bring any open window to the front for viewing or editing. Refer to the *Window Menu* section of Chapter 3, *Project Window,* for a description of the commands in the Window Menu.

Options Menu

Use the commands in the **Options** menu to select the help style, generate function prototypes, generate a `.doc` file, create a DLL project, and select whether to enable VXI*plug&play* style.

Options»FP File Format

.FP File Format allows you to select the format for the current function panel file as well as the default format for new function panel files. Two function panel file formats are available. The following table lists the features of the two formats.

FP File Format Features

Feature	CVI 5.0.1 and earlier format	CVI 5.5 and later format
Maximum Instrument Prefix Length	8	31
Maximum Function Name Length	31	79
Maximum Class Name Length	31	79
Maximum Window Name Length	31	79
Maximum Type String Length	50	80
Default Text for Output Controls	Not supported	Supported
Function Qualifiers	Not supported	Supported
Set Precision on Numeric Controls	Not supported	Supported

Options»Help Style

Help Style lets you choose the help style **New (Recommended)** or **Old (LabWindows DOS)** when you are editing context-sensitive help information of the function tree.

The new and old help styles differ significantly. The old help style maintains compatibility with function panels created in LabWindows version 2.3 or earlier. This help style uses the DOS/IBM character set so that it can display special extended ASCII characters that many older instrument drivers use. Also, the old style gives help information for the entire function panel window, not the individual function panels within a function panel window.

The new help style uses the standard Windows character set and gives help information for each individual function panel. In addition, the new help style automatically generates control name and data type information when displaying control help and automatically generates a

function prototype when displaying function help. Also, the help text editor for the new style help uses word-wrap mode.

Changing the help style only changes how the program interprets help information. If you use special extended ASCII characters in your help information and then change to the new style, you must change the help text to a Windows-compatible character set.

Use the new help style whenever possible.

Options»Transfer Window Help to Function Help

Transfer Window Help to Function Help helps you convert your function panel from old to new style. For each function panel window, the window help text is transferred to the first function unless the function already has help text.

Options»Generate Function Prototypes

Generate Function Prototypes creates an untitled `.h` window that contains prototypes for the functions in the function tree.

Options»Generate Documentation

Generate Documentation creates a window that contains a `.doc` file for the function panel file.

Options»Generate Windows Help

Generate Windows Help creates a project file (`.hpj`) and two source files (`.rtf` and `.whh`) that you can use with Microsoft Windows Help Compiler to create a Windows help file. You are prompted to choose the output language as either C or Visual Basic.

Options»Generate ODL File

Generate ODL File creates an Object Description Language (`.odl`) file for the instrument driver. The `.odl` file can be input to the `MkTypeLib` program that comes with the Microsoft OLE 2 SDK. This is useful when you create a DLL version of the instrument driver. The `MkTypeLib` program creates a type library that describes the function entry points in the DLL. Refer to the *OLE 2 Programmers Reference, Volume 2*, from Microsoft Press for information on using type libraries.

Options»Generate DEF File

Generate DEF File generates a `.def` file for the instrument driver. External compilers use the `.def` file to compile your instrument driver into a DLL. The file contains entries to export each function in the function tree.

Options»Create DLL Project

The **Create DLL Project** command creates a LabWindows/CVI project (`.prj`) file that can be used to create a dynamic link library (`.dll`) from the program file associated with the function panel (`.fp`) file. When you execute this command, you are prompted to enter a pathname for the project file. After the file is written, you are asked if you want to load the project immediately. If you do, your current project is unloaded. Refer to Chapter 3, *Compiler/Linker Issues*, of the *Measurement Studio LabWindows/CVI Programmer Reference Manual,* for more information on creating DLLs.

Options»VXIplug&play Style

VXIplug&play Style affects the contents of the DLL project that you create using the **Create DLL Project** command. If the **VXIplug&play Style** command is enabled, **Create DLL Project** adds project settings that allow the DLL, import libraries, and distribution kit you create to conform to various aspects of the VXI*plug&play* specification. You can modify all these settings using commands in the **Build** menu of the Project window. The following list describes the default settings.

- The Instrument Driver Support Only command is enabled.
- In the Create Dynamic Link Library dialog box:
  - `"_32"` is appended to the base filename of the DLL but not to the base filename of the import libraries.
  - In the Import Library Choices dialog box, the **Generate import library for all compilers** option is enabled.
- In the Type Library dialog box:
  - The **Add type library resource to DLL** option is enabled.
  - The **Include links to help file** option is enabled.
  - **Function panel file** is set to the full pathname of the `.fp` file of the current Function Tree Editor window.
- In the Change dialog box in the **Exports** section:
  - The **Export What** option is set to `Include File Symbols`.
  - The **Which Project Include Files** list contains the name of the include file associated with the `.fp` file of the current Function Tree Editor window.
- In the Create Distribution Kit dialog box:
  - The **Install Run-Time Engine** option is disabled. The instrument driver support DLL is included in the file groups instead. If you need the LabWindows/CVI Run-time Engine for the soft front panel executable, you must enable this option manually.
  - File groups are created that contain all the files that are required of a VXI*plug&play* instrument driver installation. For example, only the import libraries for Visual

C/C++ and Borland C/C++ are included, and their directory names are `msc` and `bc`. Files that you must create independently are also named in the file groups, even if they do not currently exist. These files include the following:

- A Visual Basic include file, which you can create by selecting **Options»Generate Visual Basic Include** in the Source window
- A documentation file, which you can create by selecting **Options»Generate Documentation**.
- A help file, which you can create by selecting **Options»Generate Windows Help** and the Windows Help Compiler
- A knowledge base file as defined in the VXI*plug&play* specification
- Files for a soft front panel executable (an empty file group is created for this)

- In the Advanced dialog box:
  - The **Use Custom Script** option is enabled.
  - **Script Filename** is set to `cvi\bin\vxipnp.inf`.
  - **Executable Filename** is left empty. After you create a soft front panel executable and add it to the soft front panel file group, click on the **Select** button to specify the soft front panel executable as the **Executable Filename**.
  - The **Installation Title** names are set to `<instrument prefix> Instrument Driver`.

Help Menu

You use the commands in the **Help** menu to access information about LabWindows/CVI. Refer to the *Help Menu* section of Chapter 3, *Project Window,* for a description of the commands in the Help Menu.

Function Tree Editor Examples

Creating a Function Tree with Multiple Classes

In this example you create a function tree with several nested classes. Before beginning, invoke the Function Tree Editor by selecting **File»New»Function Tree (*.fp)**.

Complete the following steps to create a new instrument and function tree.

1. Select **Create»Instrument**.
2. Enter `Function Tree Examples` as the Name and `tree` as the Prefix. Click **OK**.
3. Select **Create»Function Panel Window**.
4. Enter `Function 1` as the Name and `fun1` as the Function Name. Click **OK**.

5. Select **Create»Class**.
6. Enter `Class 1` as the Name. Click **OK**.
7. Select the line beneath the name `Class 1`.
8. Select **Create»Function Panel Window**.
9. Enter `Function 2` as the Name and `fun2` as the Function Name. Click **OK**.
10. Select **File»Save .FP File As** and save the file as `mltcls`.
11. To view the structure of the function tree as it is seen by the driver's user, select the instrument name from the **Instrument** menu.

Cutting and Pasting Functions and Panels

Frequently, you want to copy a function in a function tree and its associated function panel to a new position within the function tree.

Complete the following examples to cut and paste a function within a function tree.

1. Position the selection on the name `Function 1`.
2. Select **Edit»Cut**. The function disappears from the tree and is stored on the clipboard.
3. Position the selection on the name `Function 2`.
4. Select **Edit»Paste Above**. The function now appears under `Class 1`.

Suppose that instead of moving the function, you want to replicate it. Because the function is still in the Function Tree Editor clipboard, you can move the selection to the name `Class 1` and select **Edit»Paste Above**. The name `Function 1` reappears at the top of the tree.

Note Pasting functions and classes within the Function Tree Editor copies all items associated with the function or class, including controls and function panel help.

Using Existing Function Panels in a New Driver

Complete the following steps to copy some of the function panels from this driver to a new driver.

1. Select **File»New»Function Tree (*.fp)**. A new blank function tree window appears on the screen.
2. Select **Create»Instrument**.
3. Name the instrument `New Instrument` and type `new` in the prefix box. Click on **OK**.
4. Select **Window»Function Tree** and select the file called `mltcls`.
5. Position the selection on the item `Class 1`.
6. Select **Edit»Copy**.
7. Return to the `New Instrument` file through the **Window** menu.

8. Position the selection on the line beneath the name of the instrument.
9. Select **Edit»Paste Below**. `Class 1` and its associated functions appear in the new tree.

When you paste a class into a new tree, all information associated with the class and the functions of the class are retained.

Editing Items in the Function Tree

In this example you edit the names displayed in the function tree. You edit all the function tree items by selecting **Edit»Edit Node**.

Complete the following steps to change the name of the instrument driver and its prefix:

1. Select `New Instrument`.
2. Select **Edit»Edit Node**. The Edit Instrument Node dialog box originally used to create the instrument appears.
3. Change the name of the instrument to `Tree #2` and the prefix to `tree2`. Click on **OK**.

The changes in the instrument driver name appears at the top of the function tree in the Function Tree Editor as well as at the bottom of the window. The changes to the prefix are reflected in the Generated Code Window in each function panel.

8

Function Panel Editor

This chapter describes how to create and modify instrument driver function panels using the Function Panel Editor.

Invoking the Function Panel Editor

You can invoke the Function Panel Editor from the Function Tree Editor or from a function panel.

Invoking from the Function Tree Editor

To invoke the Function Panel Editor from the Function Tree Editor:

1. Highlight the function that corresponds to the function panel you want to edit.
2. Select **Edit»Edit Function Panel Window** on the Function Tree Editor menu bar.

You also can invoke the Function Panel Editor with the shortcut key <F8> or by double-clicking on the function name.

Invoking from a Function Panel

To edit a function panel that you are currently operating, select **Option»Edit Function Panel Window** on the Function Panel menu bar. If the current function panel is for a library that is in the **Library** menu, you cannot use the **Edit Panel** command.

Function Panel Editor Menu Bar

The following items appear on the function panel:

- The Function Panel Editor menu bar appears at the top of the screen above the function panel.
- The Instrument Name and Function Panel Name appear in the title bar of the function panel window.
- The Function Name appears in the title bar of the function panel.
- The Function Name appears with an empty argument list in the Generated Code window, below the Function Panel Editor window.

- Right-clicking a control displays a menu with the following options:
  - **Control Help**—Opens the Help Editor window.
  - **Edit Control**—Opens the Edit Control dialog box for the selected control.
  - **Change Control Type**—Opens the Change Control Type dialog box.
  - **Cut Controls**—Removes the selected control(s) from the function panel and places the control(s) on the clipboard. The contents of the clipboard stay in place when you change panels.
  - **Copy Controls**—Copies the selected control(s) and places the control(s) on the clipboard. The contents of the clipboard stay in place when you change panels.

File Menu

The **File** menu lets you create a new function tree, edit an existing function tree, save function panel information into a `.fp` and `.sub` file on disk, or add function panels to a project.

For each IVI instrument driver, a `.sub` file accompanies the `.fp` file. The `.sub` file contains the information about the instrument driver attributes. You edit this information using the attribute editor. When you save the contents of a `.fp` file, LabWindows/CVI also saves the contents of the `.sub` file automatically.

File»New, Open, Save, and Exit LabWindows/CVI

The **New**, **Open**, **Save**, and **Exit LabWindows/CVI** commands in the **File** menu of the User Interface Editor work like **New**, **Open**, **Save**, and **Exit LabWindows/CVI** commands in the Project window. For more information on these commands, refer to the *File Menu* section of Chapter 3, *Project Window*.

File»Save .FP File

The **Save** command writes the contents of the active window to disk. If you want to append a different extension, type it in after the filename. If you do not want to append an extension, enter a period after the filename.

File»Save .FP File As

The **Save As** command writes the contents of the active window to disk using a new filename you specify and changes the name of the active window to the name you specified.

File»Save Copy of .FP File

Use the **Save Copy of .FP File** command to save a copy of your `.fp` using a name you specify without changing the name of the active window.

File»Close

The **Close** command closes the active window. If you have modified the contents of the window since the last save, LabWindows/CVI prompts you to save the file to disk.

File»Save All

The **Save All** command saves all open files to disk.

File»Add .FP File to Project

The **Add .FP File to Project** command adds the `.fp` file of the current Function Panel window to the project list.

File»Add Program File to Project

The **Add Program File to Project** command adds the instrument driver program file associated with the `.fp` file of the current Function Panel window to the project list.

File»Read Only

The **Read Only** command suppresses the text editing capabilities in the current window. When you initially open a file, LabWindows/CVI disables the **Read Only** command unless the file is read only on disk.

Edit Menu

Use the commands in the **Edit** menu to modify controls, panels, and functions, add context-sensitive help information, or align and distribute objects in the Function Panel Editor.

Edit»Undo and Redo

The **Undo** command reverses your last action. Use the **Undo** command for reversing the following actions:

- moving controls
- cutting, copying, or pasting controls
- making changes using the **Control Coordinates** command.

You lose your undo information when any of the following occurs:

- You create a control.
- You edit a control, make a change, and click on the **OK** button.
- You cut the panel.

- You close the window.
- You advance to another function panel in the same window.

The **Redo** command reverses your last **Undo** command.

Edit»Cut Controls

The **Cut Controls** command removes the selected controls from the function panel and places the controls and their associated help information on the clipboard. The contents of the clipboard stay in place when you change panels.

Edit»Copy Controls

The **Copy Controls** command copies the selected controls and their associated help information to the clipboard. The contents of the clipboard stay in place when you change panels.

Edit»Paste

The **Paste** command copies objects from the clipboard and places them on a function panel window. You can paste the same object as many times as you need to.

You cannot paste a return value control on a function panel that already contains one. A function panel can contain only one return value control.

Edit»Cut Panel

The **Cut Panel** command removes the selected panel from the function panel window and places the panel, its controls, and all the associated help information on the clipboard. The contents of the clipboard stay in place when you change function panel windows.

Edit»Copy Panel

The **Copy Panel** command copies the selected panel, its controls, and all the associated help information to the clipboard. The contents of the clipboard stay in place when you change function panel windows.

Edit»Edit Control

You can modify an existing control with **Edit Control**. When you select **Edit Control**, you see the same series of dialog boxes you use to create the control. The Create Menu discusses the proper use of these dialog boxes.

Edit»Change Control Type

You can change the type of a control with **Change Control Type**. When you select **Change Control Type**, a dialog box appears that lists the available control types.

Select the desired control type from the dialog box. When you select a new control type, you see the same series of dialog boxes that you use to create the control. The *Create Menu* section gives more information about using these dialog boxes.

If you change a control type from slide to ring, or vice versa, the new control type retains the option list associated with the old control.

Edit»Edit Function

You can modify an existing function panel with **Edit Function**. When you select **Edit Function**, you see the same series of dialog boxes you use to create the panel. The *Create Menu* section discusses the proper use of these dialog boxes.

Edit»Alignment

Alignment lets you align the selected controls. Refer to the *Arrange Menu* section of the Chapter 4, *User Interface Editor Window,* for a description of this command.

Edit»Align

Edit»Align repeats your previous alignment operation. Refer to the *Arrange Menu* section of the Chapter 4, *User Interface Editor Window,* for a description of this command.

Edit»Distribution

Distribution lets you distribute the selected controls. Refer to the *Arrange Menu* section of the Chapter 4, *User Interface Editor Window,* for a description of this command.

Edit»Distribute

Distribute Vertical Centers repeats the previous distribution. Refer to the *Arrange Menu* section of the Chapter 4, *User Interface Editor Window,* for a description of this command.

Edit»Control Coordinates

Use the **Control Coordinates** command to view and set the top and left coordinates of the controls on the panel. You can use the Control Coordinates dialog box to align and move controls.

The list box displays a list of the controls in the function panel, along with the top and left coordinates and the width of each control.

Use the commands and ring controls to the right of the list box to edit the top coordinate, left coordinate, and width of each control.

The ring controls contain default values until you highlight a specific control name and click **Extract**. When you click **Extract**, LabWindows/CVI displays the coordinates for the highlighted control in the ring controls.

You can change the top and left coordinates and width of a control by highlighting a control name in the list box, entering a value in the ring controls, and clicking **Apply**.

To align or move controls, complete the following steps:

1. Use the **Extract** button to copy values into the ring control. Highlight a control name, and click **Extract** to copy the existing value of the control to the top, left, or width ring controls.
2. To apply values to other control names, place a check next to the control names whose values you want to change. Click **Apply** to copy the values.

Edit»Find

The **Find** command allows you to locate a particular text string in the function panel file. You can search for text in node names; function names; control labels; control values; item labels in ring, slide, and binary controls; message control text; and help text. When you search in help text, you cannot search in any of the other items at the same time. The search begins at the node that is currently selected.

If the **Find** command brings up a Help Editor window and you do not use the button bar, you must return to the Function Tree Editor window to continue searching throughout the panel. The **Find** command in the Help Editor window searches only within the window. You can return to the Function Tree Editor window by pressing <F7>. On the other hand, the **Find** command in the Function Panel Editor window continues searching through the entire function panel file.

Edit»Replace

The **Replace** command operates the same as the **Find** command except that you can replace the search string with another search string.

Edit»Control Help

You can add or modify context-sensitive help information for a particular control with **Control Help**.

Edit»Function Help or Window Help

You can add or modify context-sensitive help information for the entire function panel with **Function Help** or **Window Help**. **Function Help** corresponds to new style help and **Window Help** corresponds to old style help. Refer to the *Editing Help Information* section of Chapter 9, *Adding Help Information*, for more information about adding help to a function panel.

Create Menu

The **Create** menu lets you add controls, function panels, or a common control panel to a function panel window. There are nine control types in the **Create** menu: input, slide, binary, ring, numeric, output, return value, global variable, and message.

Create»Input

An **input control** accepts a variable name or value entered from the keyboard. When you select **Create»Input**, the Create Input Control dialog box appears.

The following items appear in the dialog box:

- **Control Label**—Specifies the label that appears above the control on the panel.
- **Parameter Position**—Selects the location of the control value in the function parameter list. For a control in a common control panel, **Parameter Position** specifies the control value in the parameter lists of all function panels in a function panel window. The first position is one (1).

  For a control on a function panel, **Parameter Position** specifies the control value in the parameter list after the controls in the common control panel. The first position after the controls in the common control panel is one (1). If there is no common control panel, the first position is one (1).
- **Data Type**—Selects the data type of the item entered in the input control.
- **Default Value**—Specifies the default for the input control. The default must be a valid value, a constant name, or any other valid C expression.
- **Control Width**—Specifies the width of the control in pixels. The minimum allowed is 24. The maximum allowed is 2,048. The default control width is 145 pixels.

Create»Slide

A **slide control** looks like a mechanical slide switch. A slide control specifies a parameter value depending upon the position of the cross-bar of the slide control. When you select **Create»Slide**, the Create Slide Control dialog box appears.

The following items appear in the dialog box:

- **Control Label**—Specifies the label that appears above the control on the function panel.
- **Parameter Position**—Selects the location of the control value in the function parameter list. For a control in a common control panel, **Parameter Position** specifies the control value in the parameter lists of all function panels in a function panel window. The first position is one (1).

  For a control on a function panel, **Parameter Position** specifies the control value in the parameter list after the controls in the common control panel. The first position after the controls in the common control panel is one (1). If there is no common control panel, the first position is one (1).
- **Data Type**—Selects the data type of the values in the slide control.
- **Default Value**—Selects the default for the slide control. The default must be one of the labels specified in the Edit Label/Value Pairs dialog box.
- **Label/Value Pairs**—Displays the Edit Label/Value Pairs dialog box. Use the Edit Label/Value Pairs dialog box to specify the label and value associated with each cross-bar position on the slide control. A slide control can have up to 32 labels and associated values.

Create»Binary

A binary control operates like a mechanical on/off switch. A binary control gives a parameter value one of two predefined values, depending on whether the control is in the up or down position. When you select **Create»Binary**, the Create Binary Control Dialog Box appears.

The following items appear in the Create Binary Control dialog box:

- **Control Label**—Specifies the label that appears above the control on the panel.
- **Parameter Position**—Selects the location of the control value in the function parameter list. For a control in a common control panel, **Parameter Position** specifies the control value in the parameter lists of all function panels in a function panel window. The first position is one (1).

  For a control on a function panel, **Parameter Position** specifies the control value in the parameter list after the controls in the common control panel. The first position after the controls in the common control panel is one (1). If there is no common control panel, the first position is one (1).
- **Data Type**—Selects the data type of the values in the binary control.

- **Default Value**—Selects the default for the binary control. The default can be **On** or **Off**.
- When you select the **On/Off Settings** button, the Edit On/Off Settings dialog box appears.
- **ON Text**—Specifies the label that appears next to the upper (on) position of the binary control.
- **OFF Text**—Specifies the label that appears next to the lower (off) position of the binary control.
- **ON Value**—Specifies the value, constant name, or valid C expression you want to associate with the **On** label.
- **OFF Value**—Specifies the value, constant name, or valid C expression you want to associate with the **Off** label.

Create»Ring

A ring control shows the user an option list. A ring control displays only one item at a time from its list of options. When you select **Ring** from the **Create** menu, the Create Ring Control Dialog Box appears.

The following items appear in the Create Ring Control dialog box:

- **Control Label**—Specifies the label that appears above the control on the function panel.
- **Parameter Position**—Selects the location of the control value in the function parameter list. For a control in a common control panel, **Parameter Position** specifies the control value in the parameter lists of all function panels in a function panel window. The first position is one (1).

  For a control on a function panel, **Parameter Position** specifies the control value in the parameter list after the controls in the common control panel. The first position after the controls in the common control panel is one (1). If there is no common control panel, the first position is one (1).
- **Data Type**—Selects the data type of the values in the ring control.
- **Default Value**—Selects the default for the ring control. The default must be one of the labels specified in the Edit Label/Value Pairs dialog box.
- **Control Width**—Specifies the width of the control in pixels. The minimum allowed is 24. The maximum allowed is 2,048. The default control width is 145 pixels.
- **Label/Value Pairs**—Displays the Edit Label/Value Pairs dialog box. Use this dialog box to specify the label and value associated with each entry in the ring control. A ring control can have up to 32,000 labels and associated values.

Create»Numeric

A numeric control is an input control that lets you increment or decrement a numeric value using the up and down arrows. When you select **Create»Numeric**, the Create Numeric Control Dialog Box appears.

The following items appear in the Create Numeric Control dialog box:

- **Control Label**—Specifies the label that appears above the control on the function panel.
- **Parameter Position**—Selects the location of the control value in the function parameter list. For a control in a common control panel, **Parameter Position** specifies the control value in the parameter lists of all function panels in a function panel window. The first position is one (1).

  For a control on a function panel, **Parameter Position** specifies the control value in the parameter list after the controls in the common control panel. The first position after the controls in the common control panel is one (1). If there is no common control panel, the first position is one (1).
- **Data Type**—Selects the data type of the values in the numeric control. You can choose from the following data types:

  ```
  int
  short
  char
  unsigned int
  unsigned short
  unsigned char
  double
  float
  ```

  or choose a user-defined data type for which you have specified an intrinsic type.
- **Default Value**—Selects the default for the numeric control, which must be a valid member of the value set.
- **Display Format**—Selects the output format. You can display integers, longs, and shorts in decimal, hexadecimal, octal or ASCII. You can display doubles and floats in either scientific or floating-point notation.
- **Precision**—Selects how many digits the control displays to the right of the decimal point.
- When you click on the **Value Set** button, the Edit Value Set dialog box appears.

The following items appear in the Edit Value Set dialog box:

- **Minimum**—Selects the minimum value the numeric control accepts.
- **Maximum**—Selects the maximum value the numeric control accepts.

- **Inc Value**—Selects the amount the numeric control value increments or decrements when the user presses the up or down arrows. The value in Inc Value must divide evenly into the range of the numeric control.

Create»Output

An output control displays the results of a function call. When you select **Create»Output**, the Create Output Control Dialog Box appears.

The following items appear in the Create Output Control dialog box:

- **Control Label**—Specifies the label that appears above the control on the panel.
- **Parameter Position**—Selects the location of the control value in the function parameter list. For a control in a common control panel, **Parameter Position** specifies the control value in the parameter lists of all function panels in a function panel window. The first position is one (1).

  For a control on a function panel, **Parameter Position** specifies the control value in the parameter list after the controls in the common control panel. The first position after the controls in the common control panel is one (1). If there is no common control panel, the first position is one (1).
- **Data Type**—Selects the data type of the variable or value displayed in the output control.
- **Default Value**—Specifies the default value that the control displays. You can leave this item blank.
- **Display Format**—Selects the format in which the output control displays values. You can display integers, longs, shorts, and chars in decimal, hexadecimal, octal or ASCII. You can display doubles and floats in either scientific or floating-point notation. If the data type is `char *`, `void *`, a meta data type, or an array, the display format control is not valid.
- **Control Width**—Specifies the width of the control in pixels. The minimum allowed is 24. The maximum allowed is 2,084. The default control width is 145 pixels.

Create»Return Value

A return value control displays a value returned from a function. You can use a return value control only if the function has a non-void data type. When you select **Create»Return Value**, the Create Return Value Control Dialog Box appears.

You see the following items in the Create Return Value Control dialog box:

- **Control Label**—Specifies the label that appears above the control on the function panel.
- **Data Type**—Selects the data type of the variable or value displayed in the return value control. The data type can be any data type other than an array type or a meta data type.

- **Display Format**—Selects the format in which the return value control displays values. You can display integers, longs, shorts, and chars in decimal, hexadecimal, octal or ASCII. You can display doubles and floats in either scientific or floating-point notation. If the data type is `char *` or `void *`, the display format control is not valid.
- **Control Width**—Specifies the width of the control in pixels. The minimum allowed is 24. The maximum allowed is 2,048. The default control width is 145 pixels.

Create»Global Variable

A global variable control displays the value of a global variable defined in LabWindows/CVI when users operate the function panel. When you select **Create»Global Variable**, the Create Global Variable Control Dialog Box appears.

The following items appear in the Create Global Variable Control dialog box:

- **Control Label**—Specifies the label that appears above the control on the panel.
- **Global Variable Name**—Specifies the name of the variable whose contents are shown in the global control.
- **Data Type**—Selects the data type of the item entered in the input control.
- **Display Format**—Selects the format in which the global variable control displays values. You can display integers, longs, shorts, and chars in decimal, hexadecimal, octal or ASCII. You can display doubles and floats in either scientific or floating-point notation. If the data type is `char *`, `void *`, a meta data type, or an array, the display format control is not valid.
- **Control Width**—Specifies the width of the control in pixels. The minimum allowed is 24. The maximum allowed is 2,048. The default control width is 145 pixels.

Create»Message

You can place text anywhere on the panel with a message control. This serves as an online documentation tool for panels. When you select **Create»Message**, a dialog box appears. Enter the text into the message text control and click the **OK** button. To enter a new line in the message text control, press <Ctrl-Enter>. The text appears on the panel, and you can position it like any other control.

Create»Function Panel

A function panel graphically represents a single function. Function panels can contain any of the nine different control types. A function panel can have only one return value control. The function panel window can contain more than one function panel.

Create»Common Control Panel

A common control panel contains controls that are common to all functions represented by function panels in the function panel window. Common control panels are useful only when you have multiple function panels in the function panel window. Controls on the common control panel appear as the first parameter of every function associated with a function panel window. A function panel window can contain only one common control panel. You could use a common control with an instrument driver that allows multiple instruments of the same model type to exist on a GPIB board. In this case, the common control panel can contain a control that is an index to specify which instrument is addressed.

Note In general, National Instruments recommends that you have only one function panel per window and no common control panels.

View Menu

Use the **View** menu commands to view the current instrument driver function panels or the most recently used function panels. The commands give easy access to function panels within an instrument driver.

View»Toolbar

Use the **Toolbar** command to toggle between viewing or not viewing the Function Panel window toolbar.

View»Error

If an error occurs during the execution of a function panel, you can use the **Error** command to toggle between the error message and the code in the generated code box.

View»Include File

The **Include File** command displays the include file associated with the library or instrument driver in a Source window. The include file contains all the function prototypes for the library or instrument driver.

View»Current Tree

The **Current Tree** command displays the Select Function Panel dialog box for the most recently used function panel, making it easy for you to return to the location of the current panel in the function tree.

View»Function Panel History

The **Function Panel History** command displays a scrollable list of the function panels you have used during the current LabWindows/CVI session. You can display function panels from the list as new windows, or you can overwrite the current Function Panel window.

View»Find Function Panel

When you select the **Find Function Panel** command, a dialog box appears in which you can enter the name of a function. You can enter just a substring, and the **Find Function Panel** command finds all functions that contain that substring anywhere in their names. For instance, if you enter `ctrl` and click on **OK**, a dialog box appears with a list of functions including `NewCtrl`, `SetCtrlVal`, `GetCtrlVal`, and so on.

You can use a regular expression as your search string. Refer to Table 5-1, , in Chapter 5, *Source and Interactive Execution Windows*, for a list of regular expression characters.

If a function panel exists for the function, LabWindows/CVI displays the panel. If two or more function panels exist for the function, LabWindows/CVI displays a list of the function panels.

The shortcut key for **Find Function Panel** is <Ctrl-Shift-P>.

View»Previous Function Panel

The **Previous Function Panel** command displays the previous function panel in the current Function Panel window.

View»Next Function Panel

The **Next Function Panel** command displays the next function panel in the current Function Panel window.

View»Previous Function Panel Window

The **Previous Function Panel Window** command opens the Function Panel window that precedes the current Function Panel window in the same Function Tree.

View»Next Function Panel Window

The **Next Function Panel Window** command opens the Function Panel window that follows the current Function Panel window in the same Function Tree.

The rotation order for the Function Tree is circular. If the first Function Panel window in the tree is visible on the screen, selecting **Previous Function Panel Window** displays the last

Function Panel window in the tree. If the last Function Panel window in the tree is visible, selecting **Next Function Panel Window** displays the first Function Panel window in the tree.

View»First Function Panel Window

The **First Function Panel Window** command displays the first Function Panel window in the Function Tree.

View»Last Function Panel Window

The **Last Function Panel Window** command displays the last Function Panel window in the Function Tree.

Instrument Menu

Use the commands in the **Instrument** menu to load and edit an instrument driver and to edit a function in the loaded instrument driver. The **Instrument** menu lists the loaded instrument drivers.

Instrument»Load

Select **Instrument»Load** to add a new instrument driver to the **Instrument** menu. The **Load** command operates like the **File»Open** command. When you select the **Load** command, the Load Instrument dialog box appears. Enter the appropriate information to select an existing function panel file.

Instrument»Unload

Select **Instrument»Unload** to remove one or all instrument drivers from the **Instrument** menu. When you select the **Unload** command, the Unload Instrument dialog box appears. In this dialog box, you have the following options.

- Use the mouse or the cursor keys and space bar to individually select which instrument drivers to unload.
- Select all instrument drivers by clicking the **Check All** button.
- Deselect all instrument drivers by clicking the **Check None** button.
- Click the **OK** button to unload the selected instrument drivers.
- Click the **Cancel** button to return without unloading any instrument drivers.

Instrument»Edit

The **Edit** command lets you invoke the Function Panel Editor or modify the relationship between the function panel file and its associated program file. When you select **Instrument»Edit**, the Edit Instrument Dialog Box appears.

The Edit Instrument dialog box presents the following options:

- **Show Info**—Displays the names of the current function panel file and the attached program file. It also shows whether these files are in the current project and if the program file is compiled. The attached program file contains the functions that are called when users operate the function panel.
- **Attach and Edit Source**—Searches the directory that contains the function panel file for a filename that has the same prefix as the function panel file and a `.c` extension. If the file is found, a new Source window opens with the file displayed in it, and the source file is attached to the function panel. If the file is not found, you are prompted to create a new source file and a blank Source window appears.
- **Detach Program**—Detaches the program file from the function panel.
- **Reattach Program**—Attaches a program file to a function panel. It searches the directory that contains the function panel file for a filename that has the same prefix as the function panel file and a `.obj`, `.dll`, or `.c` extension. If a file is found, the program attaches it to the function panel.
- **Edit Function Tree**—Invokes the Function Tree Editor.
- **Done**—Exits the Edit Instrument dialog box without modifying the function panel.

Tools Menu

Use the commands in the **Tools** menu to generate function definitions and declarations into your source and include files, jump to function definitions and declarations, and invoke the instrument driver developer wizard and attribute editor.

Tools»Create ActiveX Controller

Refer to the *Tools Menu* section of Chapter 3, *Project Window,* for a description of this command.

Tools»Create ActiveX Server

Refer to the *Tools Menu* section of Chapter 3, *Project Window,* for a description of this command.

Tools»Edit ActiveX Server

Refer to the *Tools Menu* section of Chapter 3, *Project Window,* for a description of this command.

Tools»Create IVI Instrument Driver

This command initiates the instrument driver developer wizard. Refer to the *Measurement Studio LabWindows/CVI Instrument Driver Developers Guide* for more information on the instrument driver developer wizard.

Tools»Edit Instrument Attributes

Edit Instrument Attributes initiates the attribute editor. Refer to the *Measurement Studio LabWindows/CVI Instrument Driver Developers Guide* for more information on the attribute editor.

Tools»Enable Auto Replace

Enable Auto Replace enables the updating of instrument driver source files to reflect changes to function names in the function tree. This option is global to LabWindows/CVI, so enabling it in one Function Tree Editor window enables it for all Function Tree Editor windows. This command is dimmed for function trees that have not yet been saved.

When this option is enabled and not dimmed, LabWindows/CVI updates the instrument driver `.c`, `.h`, and `.sub` files to reflect changes you make to function names or the instrument prefix in the Function Tree Editor window or Function Panel Editor window. When you change a function name, you are prompted for permission to update your instrument driver to reflect the new name. When you change the instrument prefix, you are prompted for permission to update your instrument driver to reflect the new prefix.

Tools»Generate Source For Function Panel

Generate Source For Function Panel generates function definitions and declarations in your driver source and header files. If a function definition already exists, LabWindows/CVI prompts you for permission to update it. You can replace, insert above or below, or skip without updating. Your choice also applies to the declaration.

Tools»Go To Definition

Go To Definition opens the driver source file and jumps to the definition of the function that is currently selected in the function tree.

Tools»Go To Declaration

Go To Declaration opens the driver include file and jumps to the declaration of the function that is currently selected in the function tree.

Window Menu

Use commands in the **Window** menu to bring any open window to the front for viewing or editing. Refer to the *Window Menu* section of Chapter 3, *Project Window*, for a description of the commands in the Window Menu.

Options Menu

Use the commands in the **Options** menu to invoke the Function Tree Editor, operate the function panel, or toggle the scroll bars.

Options»Data Types

The **Data Types** command lets you specify the names of user-defined data types. Data types you specify with the **Data Types** command appear in the Data Type Ring control on the Edit Control dialog boxes for input, slide, binary, ring, output, and global variable controls.

Note The `.h` file for the instrument driver *must* define the types that you specify with the Data Types command.

When you select **Options»Data Types,** the Edit Data Type List Dialog Box appears.

The items in the Edit Data Type List dialog box are as follows:

- **Type**—Specifies the name of a user-defined data type.
- **Intrinsic Data Type**—Allows you to associate each user defined data type with one of the intrinsic C data types that you can use in a numeric control. If you select an item other than `None`, you can use the user-defined data type as the data type for a numeric control.
- **Add**—Places the name in the **Type** control in the **Data Type** list.
- **Move Up**—Moves the selected entry up one line in the **Data Type** list.
- **Move Down**—Moves the selected entry down one line in the **Data Type** list.
- **Change**—Displays a dialog box that prompts you to change the selected entry in the **Data Type** list.
- **Delete**—Removes an entry in the **Data Type** list.

Options»Panels Movable

The **Panels Movable** command lets you specify whether panels are movable within a Function Panel Editor window. Panels are never movable in operate mode.

- **Add VISA Types**—Adds the special set of data types defined by the VISA I/O library.
- **Done**—Accepts edits to the **Data Type** list and returns to the Function Panel Editor.

Options»Toolbar

The **Toolbar** command displays a dialog box that prompts you to select which icons appear in the Function Panel Editor tool bar.

Options»Initial Control Width

With the **Initial Control Width** command, you can set the initial width for all input, output, ring, return value, and global variable controls that you create in the function panel window. The default initial control width is 145 pixels. LabWindows/CVI saves your settings between sessions.

Options»Revert to Default Panel Size

The **Revert to Default Panel Size** command sizes and positions the function panel so that it exactly fills up the default function panel window size.

Options»Toggle Scroll Bars

The **Toggle Scroll Bars** command adds or removes horizontal and vertical scroll bars from a function panel.

Options»Edit Function Tree

The **Edit Function Tree** command invokes the Function Tree Editor.

Options»Operate Function Panel

The **Operate Function Panel** command lets you operate the current function panel window.

Help Menu

You use the commands in the **Help** menu to access information about LabWindows/CVI. Refer to the *Help Menu* section of Chapter 3, *Project Window,* for a description of the commands in the Help Menu.

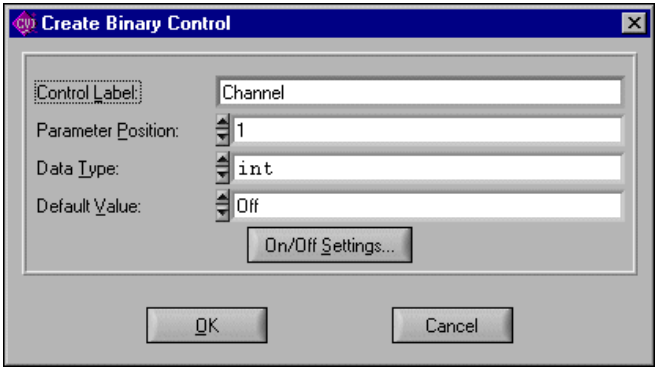
Function Panel Editor Examples

Creating a Function Window

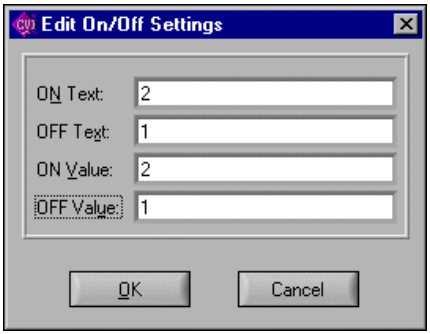
In this example, you create a function panel without writing any code. The example panel controls an oscilloscope with two channels and configures the vertical sensitivity, coupling, and invert setting of the oscilloscope.

Complete the following steps to create a new instrument and panel.

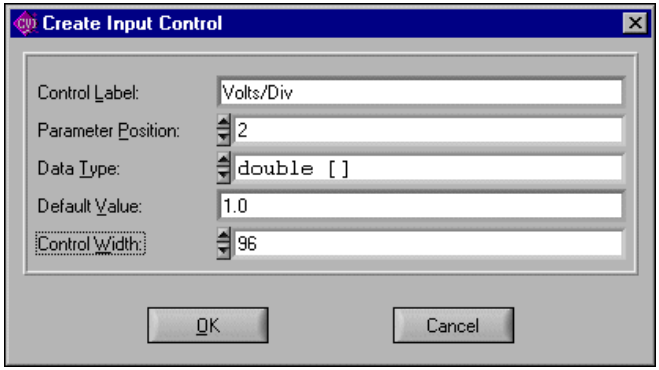
1. Select **File»New»Function Tree (*.fp)**.
2. Select **Create»Instrument**.
3. Enter `Function Panel Examples` as the Name and `panel` as the Prefix. Click on **OK**.
4. Select **Create»Function Panel Window**.
5. Enter `Configure` as the Name and `config` as the Function Name. Click **OK**.
6. Double click the `Configure` node in the function tree. A new function panel window that contains a single function panel appears on the screen. Notice that the code name of the function appears in the Generated Code window, preceded by the prefix.
7. Select **Create»Binary**.
8. Complete the Create Binary Control dialog box as shown in the following figure.



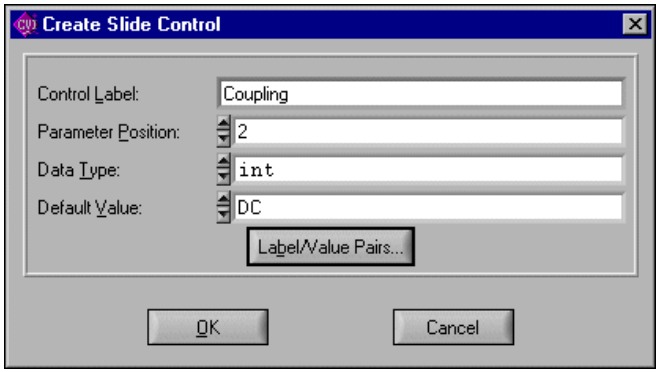
9. Click the **On/Off Setting** button and complete the Edit On/Off Settings dialog box as shown in the following figure. Click **OK** in both dialog boxes and position the control on the panel.



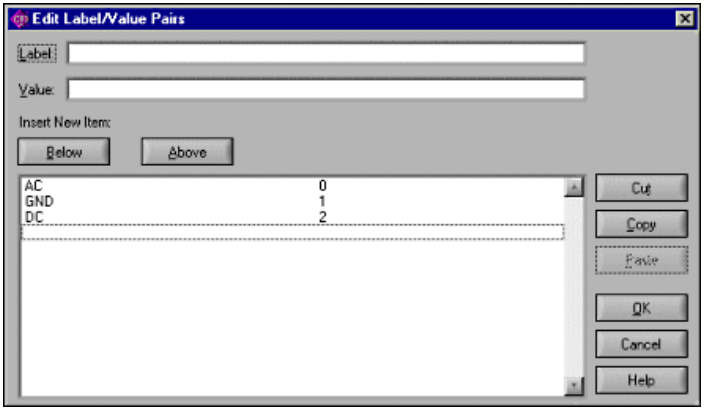
10. Select **Create»Input**.
11. Complete the Create Input Control dialog box as shown in the following figure. Click **OK** and position the control on the panel.



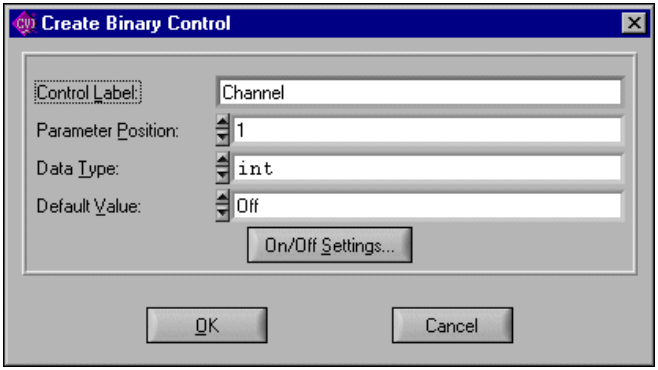
12. Select **Create»Slide**.
13. Complete the Create Slide Control dialog box as shown in the following figure.



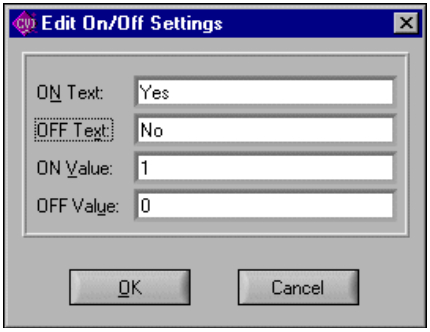
14. Click **Label/Value Pairs** and complete the Edit Label/Value Pairs dialog box as shown in the following figure. Click **OK** in both dialog boxes and position the control on the panel.



15. Select **Create»Binary**.
16. Complete the Create Binary Control dialog box as shown in the following figure.



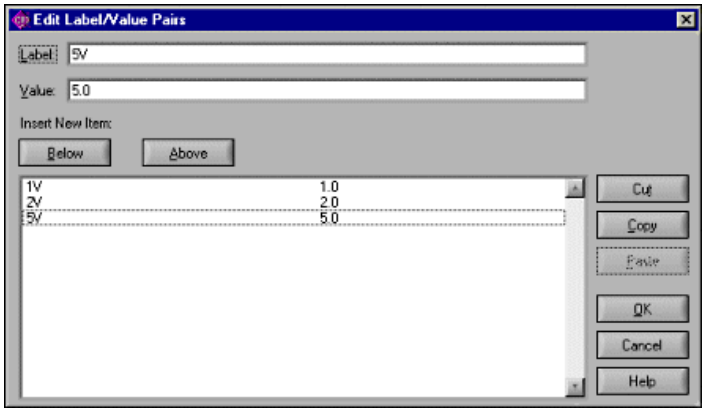
17. Click the **On/Off Settings** button and complete the Edit On/Off Settings dialog box as shown in the following figure. Click **OK** in both dialog boxes and position the control on the panel.



Changing Control Type

In this example, you change the type of the **Volts/Div** control from an input control to a slide control. Follow these steps:

1. Be sure the function panel window from the previous example is active, in edit mode. Place your cursor on the **Volts/Div** control.
2. Select **Edit»Change Control Type**.
3. In the Change Control Type dialog box, select **Slide** and click **OK**. The Edit Slide Control dialog box appears.
4. Click **Label/Value Pairs**. The Edit Label/Value Pairs dialog box appears.
5. Complete the dialog box as shown in the following figure and click **OK**.



6. Click **OK** in the Edit Slide Control dialog box to replace the **Volts/Div** input control with a slide control.

Suppose that you meant this control to be a ring control instead of a slide control. Follow these steps:

1. Place your cursor on the **Volts/Div** control.
2. Select **Edit»Change Control Type**.
3. Select **Ring**. The Edit Ring dialog box appears.
4. Click **Label/Value Pairs**, leaving all other items unchanged. The Edit Label/Value Pairs dialog box appears. Notice that the slide control label value pairs remain.
5. Click **OK**.

A ring control replaces the **Volts/Div** slide control on the function panel.

Cutting and Pasting Controls

You frequently might want to cut and paste controls. In this example, you copy controls from one panel to another. Complete the following steps to copy a control.

1. Be sure the function panel from the previous example is active and in the Edit mode. Position the selection on the **Volts/Div** control.
2. Select **Edit»Control Help** or click the secondary mouse button on the control.
3. Enter the following text in the Help Editor dialog box:

   ```
   This control specifies the volts per division setting of the
   oscilloscope.
   ```

4. Select **File»Save .FP File** and then select **File»Close** in the Help Editor dialog box.
5. With the selection still on the **Volts/Div** control, select **Edit»Copy Controls**.
6. Select **Edit»Paste**.
7. With the selection on the new control, select **Edit»Edit Control**.
8. Change the Ring Control Label to `Volts/Div 2` and the parameter position to `2`.

Notice in the Generated Code window that the `config` function now has an additional parameter, `Volts/Div 2`.

Complete the following steps to create a new function panel and copy a control to the panel.

1. Select **Option»Edit Function Tree**.
2. Create a function panel window with the following parameters. Type `New Panel` in the **Name** box and `new_panel` in the **Function Name** box.
3. Position the selection on the `Configure` node.
4. Select **Edit»Edit Function Panel Window** to return to the Configure panel.
5. Position the selection on the control **Volts/Div 2**.
6. Select **Edit»Cut Controls**.

7. Press <Ctrl-Page Down> to move to the New Panel function panel.
8. Select **Edit»Paste**.

The control appears on the panel. View the help information by selecting **Edit»Control Help**. Notice that the help information is copied with the control.

9

Adding Help Information

This chapter describes the types of help information available from an instrument driver and how you can create help information.

New Style Versus Old Style Help

LabWindows/CVI has two styles of online help for instrument drivers: new (Recommended) and old (LabWindows DOS). The old help style maintains compatibility with help information created in LabWindows version 2.3 or earlier. This help style uses the DOS/IBM character set so it can display special extended ASCII characters used by older instrument drivers.

The new help screen style uses the standard Windows character set and automatically displays the control help with control name and data type information.

There is also a difference in the type of help information that can be displayed. In either new or old style help, you can view instrument help, function class help, and control help. However, the help information for functions is displayed differently between the two styles. This difference has an effect only when you have multiple function panels on a single function panel window. In the new style, you can access function help for each function panel. In the old style, you can access the function panel window help, which describes all the functions contained in that function panel window.

National Instruments recommends that you use the new help style for all help information for instrument drivers that you create in LabWindows/CVI. Refer to *Options Menu* section in Chapter 7, *Function Tree Editor*, section for more information on new and old style help. Most of the discussion in this section assumes you are using the new style help.

Help Options

The user of an instrument driver can view the following types of help information listed in the following table.

Types of Help Information

Type of Help	Location of Help
Instrument help	Function class and function help dialog boxes
Function class help	Dialog box that appears when a user selects an instrument from the Instrument menu
Function help (New style help only)	Help menu in the function panel window menu bar
Function panel window help	Dialog box that appears when a user selects an instrument from the Instrument menu (Directly editable only in old style help. In the new style help, it is generated from the function help for each function in the window.)
Control help	Help menu in the function panel window menu bar

Editing Help Information

There are four types of help information that you can enter: instrument, class, function, and control. You can edit instrument and class help from the Function Tree Editor and function and control help from the Function Panel Editor. Each of the editors has an **Edit** menu in the menu bar. Select **Edit»Edit Help** in the Function Tree Editor to add instrument and class help. Select **Edit»Function Help** and **Edit»Control Help** in the Function Panel Editor to add function panel and control help.

Complete the following steps to add help information:

1. From either the Function Tree Editor or the Function Panel Editor, select the item to which you want to add help information.
2. Select **Edit»Edit Help**, **Edit»Function Help**, or **Edit»Control Help** in the menu bar. The Help Editor window appears.
3. Enter help in the text box.

Instrument Help

When you are viewing help information for a function class or function panel window, click the **Instrument Help** button to see help information about the instrument driver as a whole.

You can add instrument help information in the Function Tree Editor. Complete the following steps to enter the help information for the instrument.

1. In the Function Tree Editor, select the instrument node at the top of the function tree.
2. Select **Edit»Edit Help**. The Help Editor window appears. Alternatively, you can click on the instrument node with the right mouse button to display the Help Editor window.
3. Enter the help text into the Help Editor window.

Function Class Help

To display help information about a class of function panel windows, select the class in the Select Function Panel dialog box and click on the **Help** button.

Enter function class help information from the Function Tree Editor. Complete the following steps to add help information:

1. Select the class node in the function tree.
2. Select **Edit»Edit Help** in the Function Tree Editor menu bar. The Help Editor window appears. Alternatively, you can click on the class node with the right mouse button to display the Help Editor window.
3. Enter the help text into the Help Editor window.

Function Help (New Style Help Only)

When you use the new help style, you can display help information that pertains to a specific function panel by selecting **Help»Function** in the Function Panel menu bar. Alternatively, you can click on the background of the function panel with the right mouse button to display the function panel help.

When you use the new help style, you enter function panel help information from the Function Panel Editor. Complete the following steps to add function panel help:

1. Activate the function panel.
2. Select **Edit»Function Help** in the Function Panel Editor menu bar. The Help Editor window appears. Alternatively, you can click on the background of the function panel with the right mouse button to display the Help Editor window.
3. Enter the help text into the Help Editor window.

Refer to the *Options»Help Style* section in Chapter 7, *Function Tree Editor*, for more information on changing between the new and old style help modes.

Function Panel Window Help (Old Style Help Only)

When you use the old help style, you can display help information that pertains to a function panel window by selecting **Help»Window** in the Function Panel menu bar. Alternatively, you can click on the background of the function panel window with the right mouse button to display the function panel help.

When you use the old help style, you enter function panel window help information from the Function Panel Editor. Complete the following steps to add function panel window help.

1. Select **Edit»Window Help** in the Function Panel Editor menu bar. The Help Editor window appears. Alternatively, you can click on the background of the function panel window with the right mouse button to display the Help Editor window.
2. Enter the help text into the Help Editor window.

Refer to the *Options»Help Style* section in Chapter 7, *Function Tree Editor*, for more information on changing between the new and old style help modes.

Control Help

You can display help information for a specific function panel control by selecting the control and selecting **Help»Control** in the Function Panel menu bar. Alternatively, you can click on the control with the right mouse button to display the control help.

You enter control help information from the Function Panel Editor.

Complete the following steps to add help information for a function panel control.

1. Select the control.
2. Select **Edit»Control Help** in the Function Panel Editor menu bar. The Help Editor window appears. Alternatively, you can click on the control with the right mouse button to display the Help Editor window.
3. Enter the help text into the Help Editor window.

File Menu

Use the commands in the **File** menu to create a new function tree, edit an existing function tree, save function panel information into a `.fp` and `.sub` file on disk, or add function panels to a project.

For each IVI instrument driver, a `.sub` file accompanies the `.fp` file. The `.sub` file contains the information about the instrument driver attributes. You edit this information using the attribute editor. When you save the contents of a `.fp` file, LabWindows/CVI also saves the contents of the `.sub` file automatically.

File»New, Open, Save, and Exit LabWindows/CVI

The **New**, **Open**, **Save**, and **Exit LabWindows/CVI** commands in the **File** menu of the User Interface Editor work like **New**, **Open**, **Save**, and **Exit LabWindows/CVI** commands in the Project window. For more information on these commands, refer to the *File Menu* section of Chapter 3, *Project Window*.

File»Save .FP File

The **Save** command writes the contents of the active window to disk. If you want to append a different extension, type it in after the filename. If you do not want to append an extension, enter a period after the filename.

File»Save .FP File As

The **Save As** command writes the contents of the active window to disk using a new filename you specify and changes the name of the active window to the name you specified.

File»Save Copy of .FP File

Use the **Save Copy of .FP File As** command to save a copy of your `.fp` using a name you specify without changing the name of the active window.

File»Close

The **Close** command closes the active window. If you have modified the contents of the window since the last save, LabWindows/CVI prompts you to save the file to disk.

File»Save All

The **Save All** command saves all open files to disk.

File»Add .FP File To Project

The **Add .FP File To Project** command adds the `.fp` file to the project list.

File»Add Program File To Project

The **Add File To Project** command adds the file in the current window to the project list.

File»Read Only

The **Read Only** command suppresses the text editing capabilities in the current window. When you initially open a file, LabWindows/CVI disables the **Read Only** command unless the file is read only on disk.

Edit Menu

Use the commands in the **Edit** menu to edit the help text in the window.

Edit»Undo and Redo

The **Undo** command reverses your last edit action. LabWindows/CVI stores editing actions in a stack so that sequential **Undo** commands reverse a history of your edit actions.

The **Redo** command reverses your last **Undo** command. LabWindows/CVI enables the **Redo** command only when the previous action was the **Undo** command.

Edit»Cut

Cut deletes the selected text in the window and copies the text to the clipboard.

Edit»Copy

Copy copies the selected text in the window to the clipboard without deleting the selected text.

Edit»Paste

Paste inserts the contents of the clipboard into the window at the location of the cursor.

Edit»Delete

Delete discards the selected text in the window without copying it to the clipboard.

Edit»Find

Find locates a particular text string in the Help Editor window.

Edit»Replace

Replace replaces particular text in the Help Editor window with other text.

Edit»Revert

Revert returns the most recently saved version of help text to the window.

Tools Menu

Use the commands in the **Tools** menu to jump back to the function panel or function tree node that the help text in the window applies to.

Tools»Edit Function Panel

Function Panel opens the Function Panel Editor window for the function panel that contains the current help text. If the help text applies to a particular control on the function panel, the **Function Panel** command selects the control.

Tools»Edit Function Tree

Function Tree opens the Function Tree Editor window and jumps to the function tree node that contains the current help text.

Window Menu

Use commands in the **Window** menu to bring any open window to the front for viewing or editing. Refer to the *Window Menu* section in Chapter 3, *Project Window*, for command descriptions.

Help Menu

You use the commands in the **Help** menu to access information about LabWindows/CVI. Refer to the *Help Menu* section in Chapter 3, *Project Window*, for command descriptions.

Help Information Examples

Adding Help Information in the Function Tree Editor

In this example, you add instrument and function class help information to a function tree. Complete the following steps to create a new instrument and function tree and add help information to the function tree.

1. Choose **File»New»Function Tree (*.fp)**.
2. Choose **Create»Instrument**.
3. Enter `Help Information Examples` as the Name and `help` as the Prefix. Click **OK**.
4. Choose **Create»Class**.
5. Enter `Class 1` as the Name. Click **OK**.
6. Select the line beneath `Class 1`.
7. Choose **Create»Function Panel Window**.
8. Enter `Function 1` as the Name and `fun1` as the Function Name. Click **OK**.
9. The first level of help information is associated with the name of the instrument driver. Right click `Help Information Examples` to open the Help Editor window.
10. Enter the following help information.

    ```
    This driver was created to illustrate how to add help text to an
    instrument driver.
    ```

11. Select **File»Save .FP File**. Close the Help Editor window.
12. Right click `Class 1`.
13. Enter the following help information in the Help Editor window.

    ```
    An example function class. The functions in this class are the
    following:

    Function 1--The only function in the class.
    ```

14. Save the `.fp` file and close the Help Editor window.

Complete the following steps to view the help information.

1. Select **Instrument»Help Information Examples**. The Select Function Panel dialog box appears.
2. Select `Class 1` and click **Help** to open the Class Help window.
3. Click **Instrument Help** to display the Instrument Help window.
4. Click **Done** to exit the Instrument Help and Class Help windows.
5. Click **Cancel** to exit the Select Function Panel dialog box.

Adding Help Information in the Function Panel Editor

In this example, you add help information to function panels and function panel controls from the Function Panel Editor. Double-click on `Function 1` from the previous example.

Complete the following steps in the Function Panel Editor to modify the help information for the function panel.

1. Select **Edit»Function Help** from the **Edit** menu. The Help Editor window appears.
2. Enter the following help information.

   ```
   This function is the only function in Function Class.
   ```

3. Select **File»Save .FP File** then **File»Close** to save the text and remove the Help Editor window.

Help information also is associated with each of the controls in a function.

Complete the following steps to add a control to the current panel.

1. **Create»Input**.
2. Enter `Input Control` for the Control Label.
3. Click on **OK**.

Complete the following steps to add help information to the control.

1. Select the control.
2. Select **Edit»Control Help**. Alternatively, click the right mouse button on the control. The Help Editor window appears.
3. Enter the following text in the Help Editor window.
4. This control is an input control on the Function 1 function panel.
5. Select **File»Save .FP File** then **File»Close** to save the text and remove the Help Editor window.

You have now added help information to all possible locations. Select **Options»Operate Function Panel** and then view the help information for the function panel.

Copying and Pasting Help Text

In this exercise, you copy text between function panels, controls, and instruments. The clipboard retains its contents as you move between controls, function panels, and even instruments. Help text also stays with a control or function panel that is cut, copied, or pasted.

Complete the following steps to copy the help information between controls on different panels.

1. Create a new function panel window from the Function Tree Editor. Type `Function 2` in the Name box and `fun2` in the Function Name box.
2. The `Function 1` function panel should be on the screen in **Edit** mode. Double-click on `Function 1` in the Function Tree Editor.
3. Select **Create»Global Variable**.
4. Type `Status` in the Control Label box and `ibsta` in the Global Variable Name box. Leave all other items at their default settings. Click on **OK**.
5. Add the following help information to the Global Control.

```
This control displays the status of GPIB function calls.

Errors:
0         Success
non-zero  See the STATUS control on any GPIB Library function panel
```

6. Select **File»Save .FP file** then **File»Close** to save the text.
7. Select the Status control. Select **Edit»Copy Controls**.
8. <Ctrl-Page Down> to display the `Function 2` function panel.
9. Select **Edit»Paste**. The Status control appears on the function panel.
10. Select **Options»Operate Function Panel** and view the help information. Notice that the help information stays with a control when you copy that control.

Complete the following steps to copy the help text without copying the control.

1. Select **Options»Edit Function Panel** Window.
2. Select **Create»Global Variable**.
3. Complete the Create Global Variable Control dialog box as follows. Type `Error` in the Control Label box and `iberr` in the Global Variable Name box. Leave all other items at their default settings. Click on **OK**.
4. Select the Status control.
5. Select **Edit»Control Help** or click the right mouse button on the control.
6. Select all the text in the dialog box.
7. Select **Edit»Copy**.
8. Select **File»Close**.

9. Select the Error control.
10. Select **Edit»Control Help** or click the right mouse button on the control.
11. Select **Edit»Paste**. The help information appears in the window.
12. Modify the text so it reads as follows:

    ```
    This control displays the value of the GPIB global error variable.
    ```

    The control displays the value of the error only when the STATUS control is non-zero.

    ```
    Errors:
    0         Success
    non-zero  See the STATUS control on any GPIB Library function panel
    ```

In these examples, you have learned to copy or move text from one control to another. Use the same methods to copy and move help text between various locations. For example, copying and moving panel, instrument, window, and control help within an instrument driver or across instrument drivers.

10

Variables and Watch Windows

Variables Windows

You use the Variables window to inspect and modify the values of program variables. You can invoke this window when no program is running or when a program is suspended at a breakpoint.

The Variables window shows the names and types of all variables, including arrays and strings. The current values of numeric scalars, values and contents of pointers, and string contents appear in the Variables window.

Note When strings appear in ASCII format, there is *no* visual distinction between a space (ASCII 32) and a NUL byte (ASCII 0). You can see the difference by displaying the string in decimal format.

To view the Variables window, select **Window»Variables** in the active LabWindows/CVI window. You also can invoke the Variables window for the currently highlighted variable from a Source or Function Panel window with the **Run»View Variable Value** command in the Source window or the **Code»View Variable Value** command in the Function Panel window.

The Variables window shows all currently defined variables in LabWindows/CVI. LabWindows/CVI updates variables in this window at each breakpoint. The vertical bars separate the window into three scrollable fields: name, value, and variable type. You can change the width of the fields by dragging the vertical bars with the mouse. The window is also divided into two horizontal sections: the Global subwindow and the function subwindow.

The Global subwindow displays the following variables:

- Project globals that include all global variables not declared as `static`
- Interactive Execution window variables declared in the Interactive Execution window
- Global variables declared as `static`

The function subwindow displays function parameters and local variables from currently active functions. The variable list for each function appears in a different section. For any given function, the Variables window lists formal parameters first followed by local variables. Formal parameters appear in italics.

The following icons appear to the left of certain variables.

The variable on this line is the starting pointer to a block of defined data such as an array, string, or structure. Click on this icon or select **View»Expand Variable** to expand the variable so that you can see each element or member. For more information, refer to the *View»Expand Variable* section.

The variable on this line is the starting pointer to a block of defined data that appears in expanded form. Click on this icon or select **View»Close Variable** to close the variable so that you see only the starting pointer. For more information, refer to the *View»Expand Variable* section.

The variable on this line is a member of a structure that is a parent pointer to another structure of the same type. Click on this icon or select **View»Follow Pointer Chain** to replace the current structure with the child structure that the pointer references. For more information, refer to the *View»Follow Pointer Chain* section.

The variable on this line is a child structure in a chain. The pointer to its parent structure does not appear. Click on this icon or select **View»Retrace Pointer Chain** to replace the current structure with its parent. For more information, refer to the *View»Retrace Pointer Chain*.

Watch Window

The Watch window is similar in nature to the Variables window except that you can select your own set of variables and expressions to view in the Watch window. By default, LabWindows/CVI updates variables and expressions in the Watch window at each breakpoint, but you also can set them to update continuously and cause a breakpoint when their values change. To activate the Watch window, select **Window»Watch** in the active LabWindows/CVI window.

Select Watch window variables from the Variables window using the **Options»Add Watch Expression** command. The **Add Watch Expression** command opens the Add/Edit Watch Expression dialog box.

File Menu

File»New, Open, Save, and Exit LabWindows/CVI

The **New**, **Open**, **Save**, and **Exit LabWindows/CVI** commands in the **File** menu of the User Interface Editor work like **New**, **Open**, **Save**, and **Exit LabWindows/CVI** commands in the Project window. For more information on these commands, refer to the *File Menu* section of Chapter 3, *Project Window*.

File»Output

The **Output** command writes the contents of the window to an ASCII file on disk. When you select **Output**, a dialog box appears prompting you to specify the name of the file.

File»Hide

The **Hide** command visually closes a window while retaining the contents in memory.

File»Save All

The **Save All** command saves all open files to disk.

File»Most Recently Closed Files

For your reference, two lists appear in the **File** menu.

- A list of the four most recently closed files, other than project files
- A list of the four most recently closed project files

File»Exit LabWindows/CVI

The **Exit LabWindows/CVI** command closes the current LabWindows/CVI session. If you have modified any open files since the last save or if any windows contain unnamed files, LabWindows/CVI prompts you to save them to disk.

Edit Menu for the Variables Window

This section contains a detailed description of the **Edit** menu for the Variables window.

Edit»Edit Value

You can change the value of a variable with the **Edit Value** command. You can invoke the **Edit Value** command with the mouse by double-clicking on the variable name. When the dialog box appears, type in the new value.

The value that you enter in the Edit dialog box depends on the type and display format of the variable, as the following instructions demonstrate:

- Edit integers and longs in the format in which they appear.
- Edit real numbers in either scientific or floating-point format, regardless of the display format.
- Edit individual array elements by expanding the array using the **View»Expand Variable** command.
- Edit individual bytes of a string by expanding the string using the **View»Expand Variable** command. The bytes appear in the integer format you specify in the **Format** menu.

Edit»Find

The **Find** command invokes the **Find** dialog box.

Enter the text you want to find in the **Find What** text box. If you select text on a single line before you execute the **Find** command, the selected text appears in the **Find What** text box. Otherwise, the text you last searched for appears in the box. You can access a history of selections for the **Find What** text box by clicking the arrow to the right of the **Find What** text box or by using the up or down arrow keys on your keyboard.

- **Case Sensitive**—Finds only the instances of the specified text that match exactly. For example, if `CHR` is the specified text, the **Case Sensitive** option finds `CHR` but not `Chr`.
- **Whole Word**—Finds the specified text only when the characters that surround it are spaces, punctuation marks, or other characters not considered parts of a word. LabWindows/CVI treats the characters A through Z, a through z, 0 through 9, and underscore (_) as parts of a word.
- **Wrap**—Specifies to continue searching from the beginning of the window once the end of the window has been reached.
- **Regular Expression**—If you select this option, LabWindows/CVI treats certain characters in the **Find What** text box as regular expression characters instead of literal characters. Table 5-1, , in Chapter 5, *Source and Interactive Execution Windows*, describes the regular expression characters.
- **Name**—Activate this option to include the variable name field of the Variables/Watch window in the search.
- **Value**—Activate this option to include the value field of the Variables/Watch window in the search.
- **Type**—Activate this option to include the variable type field of the Variables/Watch window in the search.
- **Button Bar**—Use this option to enable or disable the built-in dialog box for interactive searching.

Find Prev and **Find Next** search for the closest previous or next occurrence of the specified text. **Stop** terminates the search, leaving the highlight on the current line. **Return** terminates the search, moving the highlight to where you initiated the search.

The search hot keys remain active even if you disable the Button Bar. The search hot-keys in the Variables and Watch windows are the same as the search hot-keys in Source windows. Use the **Keyboard Help** command in the **Options** menu of a Source window for a list of the search hot-keys.

Edit»Next Scope

In the function subwindow, **Next Scope** highlights the function that called the current function. In the Global subwindow, **Next Scope** highlights the next module. This command is not available in the Watch window.

Edit»Previous Scope

In the function subwindow, **Previous Scope** highlights the function that the current function called directly. In the Global subwindow, **Previous Scope** highlights the previous module. This command is not available in the Watch window.

Edit Menu for the Watch Window

Edit»Edit Value

The **Edit Value** command operates the same way as it does in the Variables window. Refer to the *Edit Menu for the Variables Window* section for more information on this command.

Edit»Add/Edit Watch Expression

The **Add/Edit Watch Expression** dialog box has the following options:

- **Variable/Expression**—Contains the variable or expression to place in the Watch window.
- **Scope**—Corresponds to whether the variable or expression variables are global to the project, global to a file, local to a function, or global to the Interactive Execution window.
- **Executable/DLL**—Indicates the executable or DLL to which the watch expression applies. The default value for the control is the debug executable or DLL name for the active project. When you start debugging a project, LabWindows/CVI changes an empty string to the name of the debug executable or DLL for the current project. The menu ring to the right of the control contains the names of all debuggable executables and DLLs in the workspace. If you want the watch expression to apply to a DLL that is not in the workspace, you must supply the name of the DLL. Enter the filename and extension,

without a directory path, such as `mydll.dll`. To set a watch expression for a DLL, it is easiest to first set a breakpoint in a DLL source file. Once the DLL has been loaded and program execution suspends, select the DLL name from the menu ring.

- **File**—Name of the file that defines the variable or expression variables if they are global to a file or local to a function.
- **Function**—Name of the function that defines the variable or expression variables if they are local to a function.
- **Update Display Continuously**—Causes the variable or expression to be evaluated and updated on the Watch window between each statement in your program while the program is running.
- **Break When Value Changes**—Suspends the program when the value of the variable or expression changes.
- **Replace**—Replaces the previous attributes of the current variable or expression of the same name in the Watch window with the current attributes of the dialog box. **Replace** is available only when you invoke the dialog box from the Watch window.
- **Add**—Inserts the variable or expression into the Watch window.
- **Cancel**—Aborts the operation.

You can add watch expressions to the Watch window directly from a Source window or a Function Panel window. To add a watch expression from a Source window, highlight the expression and select **Run»Add Watch Expression**. To add a watch expression from a Function Panel window, highlight the expression and select **Code»Add Watch Expression**.

Edit»Delete Watch Expression

Delete Watch Expression removes the selected watch variable/expression from the Watch window. This command is not available in the Variables window

Edit»Find

The **Find** command operates the same as it does in the Variables window. Refer to *Edit Menu for the Variables Window* section for more information on this command.

View Menu

This section contains a detailed description of the **View** menu for the Variables and Watch windows.

To use one of these commands, select a particular array or string by clicking on it with the mouse or using the up and down arrow keys and then access the command from the **View** menu.

View»Expand Variable

The Variables and Watch windows can display arrays, strings, and structures in closed form or expanded form. In closed form, you see only the name and address of the aggregate variable next to the triangle icon.

In expanded form, the icon changes to a circle, and you see the individual elements and their values.

The **Expand Variable** command expands a currently closed aggregate variable so you can see its contents. Clicking on the triangle icon has the same effect as selecting **View»Expand Variable**.

View»Close Variable

Refer to the *View»Expand Variable* section for a discussion of expanded and closed variables.

The **View»Close Variable** command closes the currently expanded aggregate variable so you can see its name and starting address. Clicking on the circle icon has the same effect as selecting **View»Close Variable**.

View»Follow Pointer Chain

Use **Follow Pointer Chain** to examine complex pointer-linked structures such as linked lists and trees. If a pointer is a member of a structure and points to a structure of the same type, **Follow Pointer Chain** replaces the current structure with the child structure that the pointer references.

Clicking on the right arrow icon or selecting **Follow Pointer Chain** replaces the current structure with the child structure that the pointer references.

View»Retrace Pointer Chain

Retrace Pointer Chain replaces the current structure with its parent. Notice the presence of the left arrow icon after selecting **Follow Pointer Chain** in Child Structure Pointer in a Chain. This indicates that the structure `hquework->begin->next` is a child structure in a chain. Clicking on the left arrow icon or selecting **Retrace Pointer Chain** causes the variable display to revert back to Parent Structure Pointer in a Chain.

Note **Retrace Pointer Chain** is valid *only* when you displayed the current structure with **Follow Pointer Chain**.

View»Go To Execution Position

This command is valid only in the Variables window. The **Go To Execution Position** command is available only when the currently highlighted item is a function name or the name of a formal parameter or local variable. The command opens the Source window that contains the call to the function in which your program suspended execution and highlights the function call. To execute the **Go To Execution Position** command, you can double-click on the function name or press <Ctrl-P>.

View»Go To Definition

This command is valid only in the Variables window. The **Go To Definition** command opens the Source window that contains the definition of the currently selected function or variable and highlights the definition.

View»Source Code Browser

This command operates the same way as the **Tools»Source Code Browser** command in the Project Window. Refer to the *Tools Menu* section of Chapter 3, *Project Window,* for more information.

View»Array Display

The **Array Display** command invokes the Array Display window for the currently highlighted array. To invoke the Array Display window, double-click on an array variable or press <F4>. Refer to Chapter 11, *Array and String Display Windows*, for more information.

View»String Display

The **String Display** command invokes the String Display window for the currently highlighted string. To invoke the String Display window, double-click on a string variable or press <Shift-F4>. Refer to Chapter 11, *Array and String Display Windows*, for more information.

View»Memory Display

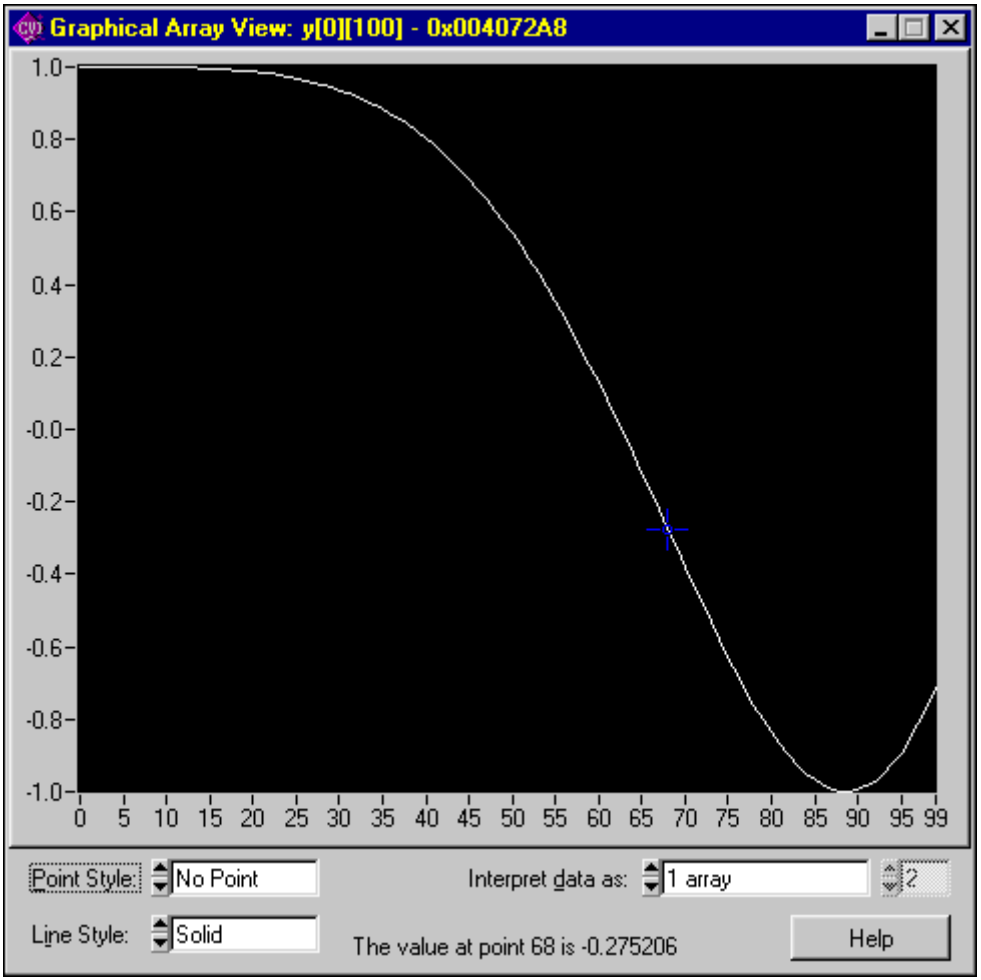
The **Memory Display** command displays the currently highlighted item in the memory display. If the currently highlighted item is a pointer variable, the memory pointed to by the pointer is displayed in the memory display. If the currently highlighted item is not a pointer, the address of the highlighted variable is displayed in the memory display.

View»Graphical Array View

You can use the **Graphical Array View** command to view the values of your arrays in a graph. You can view arrays in a graph only while you are debugging and only with 1D and 2D arrays. From the Variables window or Watch window use the **View»Graphical Array View** command to invoke the Graphical Array view for the currently highlighted array.

1D Arrays

The following figure shows a Graphical Array View for a 1D array.

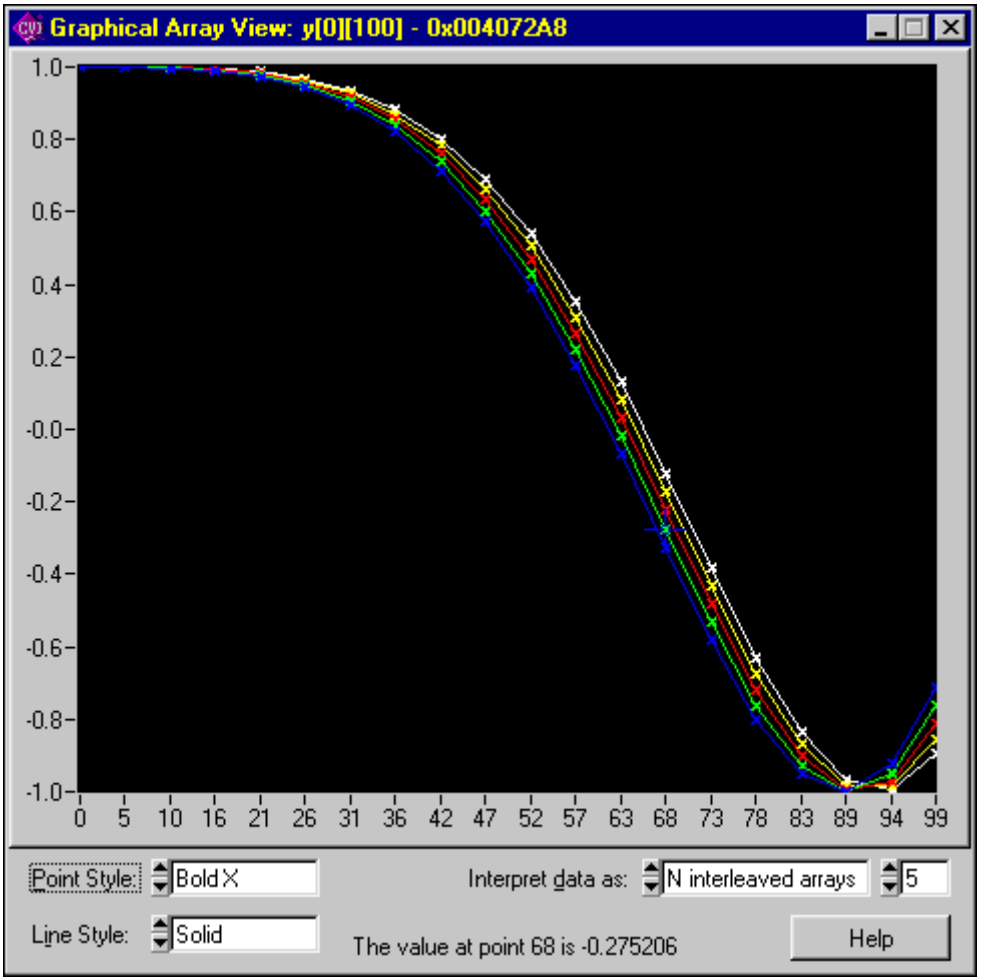


For a 1D array, the Graphical Array View shows a single plot. Your cursor changes to a crosshair pointer in the graph. To find the value of a point, move your pointer over any point

in the graph. When values change during debugging, the graph auto scales to fit the updated values.

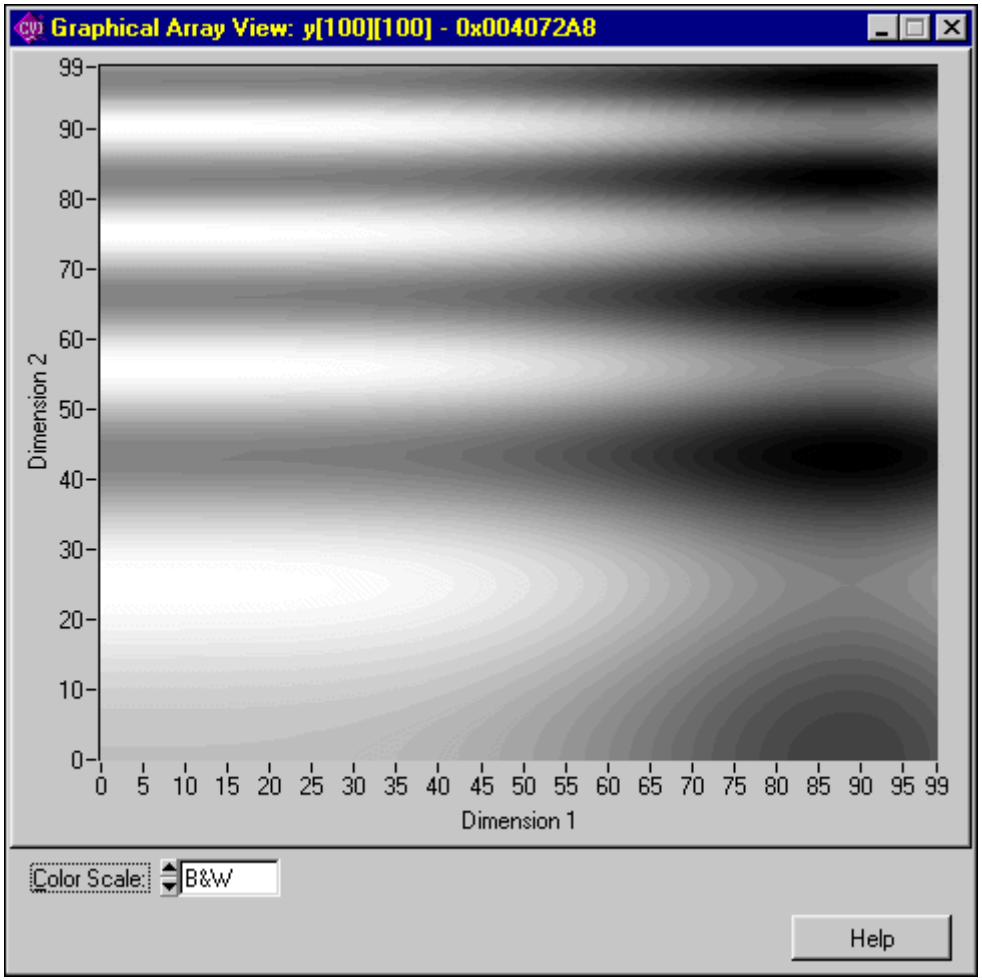
You can customize the appearance of the graph with the following options:

- **Point Style**—Selects the style in which points appear.
- **Line Style**—Selects the style in which lines appear.
- **Interpret data as**—Displays the data as 1 array or as an interleaved array. Selecting **N Interleaved Arrays** displays the data as contiguous sets of points. The maximum number of points you can select is half the number of elements in your array. The following figure illustrates an interleaved 1D array.



2D Arrays

The following figure shows a Graphical Array View for a 2D array.



A graph for a 2D array is an intensity plot. The different shades of gray represent the magnitude of the points. Darker shades represent lower values, and lighter shades represent higher values. The color scale uses the standard spectrum. The following colors are organized from highest value to lowest value.

1. White
2. Red
3. Yellow

4. Green
5. Cyan
6. Blue
7. Magenta
8. Black

The **Dimension 1** axis represents the first dimension of the array. The **Dimension 2** axis represents the second dimension of the array.

Customize the appearance of the graph by using the **Color Scale** option, which selects different color scales.

Format Menu

This section contains a detailed description of the **Format** menu for the Variables and Watch windows.

Use the commands in the **Format** menu to choose the format the Variables window uses to display numbers. You can change the format for an individual variable as well as the default formats for all variables. The first five items in the menu specify the available formats for displaying individual integers in the Variables and Watch windows. You can display integers in decimal, hexadecimal, octal, binary, or ASCII format. The next two items in the **Format** menu specify the formats available for displaying individual real numbers. Real numbers appear in either floating-point or scientific notation. The last item, **Preferences**, sets the default formats for all integers and all real numbers.

Run Menu

The **Run** menu contains the following subset of the commands that appear in the **Run** menu of the Source window.

- **Debug**
- **Continue**
- **Step Over**
- **Step Into**
- **Finish Function**
- **Terminate Execution**
- **Break at First Statement**
- **Breakpoints**
- **Threads**

Refer to the *Run Menu* section in Chapter 5, *Source and Interactive Execution Windows*, for descriptions of each of these commands.

Window Menu

The **Window** menu in the Variables and Watch windows operates the same way as it does in the Project window. Refer to the *Window Menu* section in Chapter 3, *Project Window*, for command descriptions.

Options Menu

This section contains a detailed description of the **Options** menu for the Variables and Watch windows.

To use one of these commands, select a particular variable by clicking on it with the mouse or using the up and down arrow keys then access the command from the **Options** menu.

Options»Variable Size

The **Variable Size** command displays the number of bytes the variable consumes. If you declare the variable as a buffer, the variable size is the total size of the buffer. If you declare the variable as a pointer, the **Variable Size** command displays the number of bytes the pointer itself consumes and the number of bytes in the object that the pointer references. For example, if your code contains the following declaration:

```
static double y_array [4];
```

Variable Size displays a variable size of 32 bytes for `y_array`.

Assume your code defines `dblPtr` as follows:

```
static double *dblPtr;
dblPtr = malloc (2 * sizeof(double));
```

Variable Size displays a variable size of 4 bytes for `dblPtr`, pointing to 16 bytes (2 elements).

Options»Interpret As

The **Interpret As** command displays a variable as if it were another type. Selecting a type from the Available Types dialog box displays the variable as the new type.

If **Interpret As** does not offer the exact type you want, you can use a watch expression.

Options»Estimate Number of Elements

The Variables window normally cannot expand variables for which LabWindows/CVI does not have user protection information. You can use this command to estimate the number of elements for a variable. Once you have estimated the number of elements for the variable, you can view the elements in the Variables window.

Options»Add Watch Expression (Variables Window Only)

Add Watch Expression invokes the Add/Edit Watch Expression dialog box from the Variables window. The *Edit Menu* section describes this dialog box.

Help Menu

The **Help** menu for the Variables and Watch windows works the same way as the **Help** menu in the Project window. Refer to the *Help Menu* section of Chapter 3, *Project Window*, for information on the **Help** menu.

11

Array and String Display Windows

This chapter describes the Array and String Display windows. Use these windows to inspect and modify the contents of a single array or string during a breakpoint.

Note When strings appear in ASCII format, there is no visual distinction between a space (ASCII 32) and a NUL byte (ASCII 0). You can see the difference by displaying the string in decimal format. In the String Display, you can see beyond the NUL byte by selecting **Options»Display Entire Buffer**.

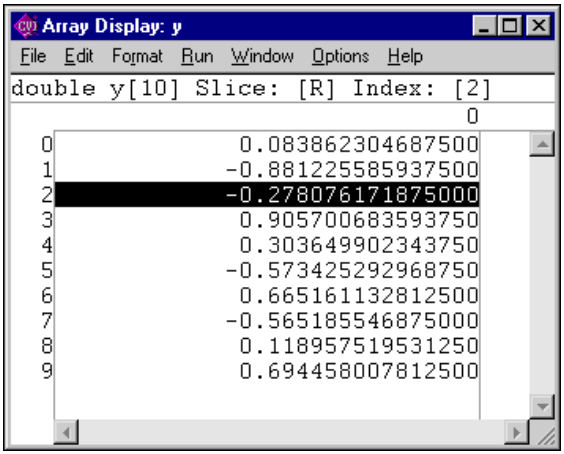
Array Display Window

You can use the Array Display window to view and edit the contents of an array or string during a breakpoint.

From the Variables window, use the **View»Array Display** command to invoke the Array Display window for the currently highlighted array. You also can double-click on an array to invoke the Array Display window.

Select **Run»View Variable Value** in a Source window or **Code»View Variable Value** in a Function Panel window to invoke the Array Display window when the name of an array variable is under the keyboard cursor or is in the active function panel control.

The following figure shows the Array Display window for a single-dimensional array.

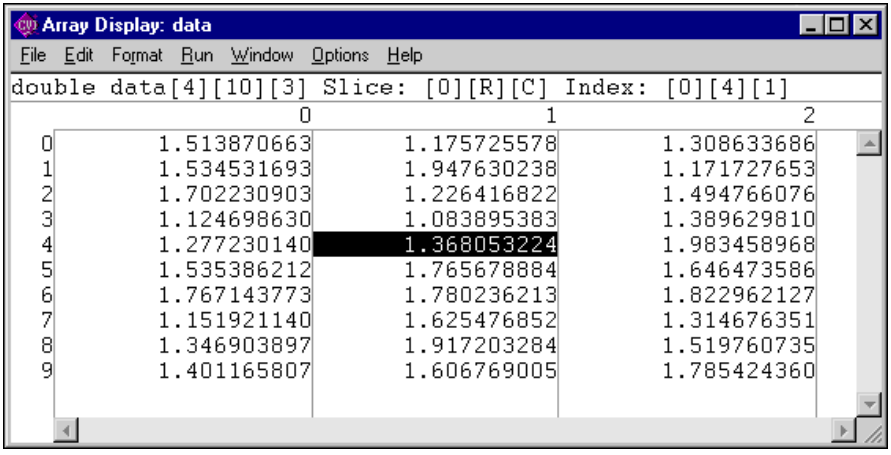


The Slice indicator shows the dimension that appears. You can display a single-dimensional array by row [R] or column [C] using the **Options»Reset Indices** command.

The Index indicator shows the currently selected element.

Multi-Dimensional Arrays

For an array with two or more dimensions, you can specify two dimensions as the rows and columns of the display. You also can specify constant values to use to fix the other dimensions. Use the **Options»Reset Indices** command to specify which plane of the array to display. The following figure shows the Array Display for a three-dimensional array.



The Array Display window shows a two-dimensional view. By default, the next-to-last dimension appears as rows, the last dimension appears as columns, and the indices of the other dimensions remain constant at 0. Select **Options»Reset Indices** to specify the dimensions you want to display as rows and columns and set the other dimensions to constant values. When you select **Reset Indices** for a three-dimensional array, the Reset Indices dialog box appears.

The dialog box shows the size and display index for each array dimension. The letter `R` indicates the dimension displayed as rows, and the letter `C` indicates the dimension displayed as columns. The indices for the remaining dimensions, those dimensions not specified as either row or column, remain constant at the specified value.

If you enter an invalid character, such as a non-alphanumeric character or any alphabetic character besides `R`, `r`, `C`, or `c`, an error message appears. Likewise, if you enter an index out of the range of a dimension, an error message appears. Press <Enter> to remove the error message. If you want to close the Reset Indices dialog box without changing the indices, click on **Cancel**.

String Display Window

You can use the String Display window to view and edit the contents of a string variable or string array during a breakpoint.

When strings appear in ASCII format, there is no visual distinction between a space (ASCII 32) and a NUL byte (ASCII 0). You can see the difference by displaying the string in decimal format. In the String Display, you can see beyond the NUL byte by selecting **Options»Display Entire Buffer**.

From the Variables window, select **View»String Display** to invoke the String Display window for the currently highlighted string variable. Double-click on a string to invoke the String Display window.

Select **Run»View Variable Value** in a Source window or select **Code»View Variable Value** in a Function Panel window to invoke the String Display window when the name of a string variable is under the keyboard cursor or is in the active function panel control.

Multi-Dimensional String Array

Use the **Reset Indices** command to specify which index of a multi-dimensional string array to use as rows in the String Display window. LabWindows/CVI disables **Reset Indices** when you view a single string variable or a one-dimensional string array. For a string array of two or more dimensions, you can specify which index to use for the rows of the display. The other dimensions remain constant at indices that you specify. When you select **Reset Indices**, the Reset Indices dialog box appears.

The dialog box shows the size and display index for each array dimension. The letter "R" indicates the dimension displayed as rows. The indices for the remaining dimensions remain constant at the specified values.

If you enter an invalid character, or any alphabetic character besides R or r, or an invalid index, an error message appears.

File Menu

This section contains a detailed description of the **File** menu for the Array and String Display windows.

File»New, Open, Save, and Exit LabWindows/CVI

The **New**, **Open**, **Save**, and **Exit LabWindows/CVI** commands in the **File** menu of the User Interface Editor work like **New**, **Open**, **Save**, and **Exit LabWindows/CVI** commands in the Project window. For more information on these commands, refer to the *File Menu* section of Chapter 3, *Project Window*.

File»Output

The **Output** command writes the contents of the window to an ASCII or binary data file on disk. When you select **Output**, a dialog box appears prompting you to specify the name of the file.

File»Input

This command is valid only in the Array Display window. Use the **Input** command to select an ASCII or binary data file on disk to replace the currently viewed array in memory.

File»Close

The **Close** command closes the window. If you have modified the contents of the window since the last save, LabWindows/CVI prompts you to save the file to disk.

File»Save All

The **Save All** command saves all open files to disk.

File»Most Recently Closed Files

For your reference, two lists appear in the **File** menu.

- A list of the four most recently closed files, other than project files
- A list of the four most recently closed project files

Edit Menu for the Array Display Window

This section contains a detailed description of the **Edit** menu for the Array Display window.

Edit»Edit Value

The **Edit Value** command in the Array Display window invokes a dialog box that you can use to change the value of the selected array element.

Edit»Find

The **Find** command invokes the Find dialog box.

The **Find** command operates the same way as it does in the Variables window, but with fewer options. Refer to the *Edit Menu for the Variables Window* section in Chapter 10, *Variables and Watch Windows*, for information on how to use options in the Find dialog box.

Edit»Goto

The **Goto** command moves the highlight to a particular location in the current string or array plane. When you execute the **Goto** command, a dialog box appears where you can enter the row and column number of the desired location. For a single string, you specify only the column.

Edit Menu for the String Display Window

This section contains a detailed description of the **Edit** menu for the String Display window.

Edit»Edit Character

Use the **Edit Character** command in the String Display window to change one character at a time.

Edit»Edit Mode

The **Edit Mode** command places the String Display window in edit mode so you can directly edit the string from the keyboard. This mode is valid only when you select the ASCII display format from the **Format** menu. Also, you can edit one character at a time using the **Options»Edit Character** command.

Edit»Overwrite

Use the **Overwrite** command in the String Display window to toggle between the overwrite and insert modes of editing. The **Overwrite** command has no effect unless you activate the **Edit Mode** command.

Edit»Find

The **Find** command invokes the Find dialog box.

The **Find** command operates the same way as it does in the Variables window, but with fewer options. Refer to the *Edit Menu for the Variables Window* section in Chapter 10, *Variables and Watch Windows*, for information on how to use options in the Find dialog box.

Edit»Goto

The **Goto** command moves the highlight to a particular location in the current string or array plane. When you execute the **Goto** command, a dialog box appears where you can enter the row and column number of the desired location. For a single string, you specify only the column.

View Menu for the Array Display Window

View»Source Code Browser

This command operates the same way as **Tools»Source Code Browser** in the Project Window. Refer to the *Tools Menu* section in Chapter 3, *Project Window*, for more information.

View»String Display

This command operates the same way as **View»String Display** in the Variables and Watch Windows. Refer to the *View Menu* section of Chapter 10, *Variables and Watch Windows*, for more information.

View»Memory Display

This command operates the same way as **View»Memory Display** in the Variables and Watch Windows. Refer to the *View Menu* section of Chapter 10, *Variables and Watch Windows*, for more information.

View»Graphical Array View

This command operates the same way as **View»Graphical Array View** in the Variables and Watch Windows. Refer to the *View Menu* section of Chapter 10, *Variables and Watch Windows*, for more information.

Format Menu

This section contains a detailed description of the **Format** menu for the Array and String Display windows.

Use the commands in the **Format** menu to choose the format the Array or String Display window uses to display numbers. You can display integers in decimal, hexadecimal, octal, binary, or ASCII format. You can display real arrays in either floating-point or scientific notation.

Run Menu

The **Run** menu contains the following subset of the commands that appear in the **Run** menu of the Source window. Refer to the *Run Menu* section of Chapter 5, *Source and Interactive Execution Windows*, for more information.

- **Run**
- **Continue**
- **Step Over**
- **Step Into**
- **Finish Function**
- **Terminate Execution**
- **Break at First Statement**
- **Breakpoints**
- **Threads**

Window Menu

Use commands in the **Window** menu to bring any open window to the front for viewing or editing. Refer to the *Window Menu* section in Chapter 3, *Project Window*, for command descriptions.

Options Menu

This section contains a detailed description of the **Options** menu for the Array and String Display windows.

Options»Reset Indices

Use **Reset Indices** in the Array Display window to set which array dimension appears as rows and which array dimension is displayed as columns.

Use **Reset Indices** in the String Display window to set which string array dimension appears as rows.

Options»Display Entire Buffer

This command is valid only in the String Display window. By default, the String Display window displays only the characters preceding the first ASCII NUL. To see characters beyond the NUL, select **Options»Display Entire Buffer**.

Help Menu

You use the commands in the **Help** menu to access information about LabWindows/CVI. Refer to the *Help Menu* section of Chapter 3, *Project Window*, for information on the **Help** menu.

A

Source Window Keyboard Commands

The following table can help you quickly identify common Source window keyboard commands that are not in the menus.

Table A-1. Keyboard Help

Category	Action	Shortcut Key(s)
Finding/Searching	Find again (up)	<Ctrl-F3>
	Find again (down)	<F3>
	Use selected text as search string	<Ctrl-Shift-F3>
	Replace selected text	<Ctrl-F11>
	Replace selected text and find again	<F11>
	Use selected text as replace string	<Ctrl-Shift-F11>
Windowing	Next window	<Ctrl-F6>
	Previous window	<Crtl-Shift-F6>
	Switch subwindows	<F6>

Table A-1. Keyboard Help (Continued)

Category	Action	Shortcut Key(s)
Editing	Change text selection mode	<Ctrl-Ins>
	Toggle insert/overwrite mode	<Ins>
	Delete to end of line	<Ctrl-D>
	Backspace to beginning of word	<Ctrl-Shift-BkSp>
	Cut line to Clipboard	<Ctrl-Y>
	Insert a new line above	<Shift-Enter>
	Insert a new line below	<Ctrl-Enter>
	Select text	<Shift-*arrow key*>
	Remove text selection	<Esc>
Cursor Movement	Up 1 line	<Up arrow>
	Down 1 line	<Down arrow>
	Left 1 column	<Left arrow>
	Right 1 column	<Right arrow>
	Scroll up 1 line	<Ctrl-Up arrow>
	Scroll down 1 line	<Ctrl-Down arrow>
	Left 1 word	<Ctrl-Left arrow>
	Right 1 word	<Ctrl-Right arrow>
	Top of window	<Ctrl-PgUp>
	Bottom of window	<Ctrl-PgDown>
	Beginning of line	<Home>
	End of line	<End>
	Move up 1 page	<PgUp>
	Move down 1 page	<PgDown>
	Top of file	<Ctrl-Home>
	Bottom of file	<Ctrl-End>

B

Technical Support Resources

Web Support

National Instruments Web support is your first stop for help in solving installation, configuration, and application problems and questions. Online problem-solving and diagnostic resources include frequently asked questions, knowledge bases, product-specific troubleshooting wizards, manuals, drivers, software updates, and more. Web support is available through the Technical Support section of `ni.com`.

NI Developer Zone

The NI Developer Zone at `ni.com/zone` is the essential resource for building measurement and automation systems. At the NI Developer Zone, you can easily access the latest example programs, system configurators, tutorials, technical news, as well as a community of developers ready to share their own techniques.

Customer Education

National Instruments provides a number of alternatives to satisfy your training needs, from self-paced tutorials, videos, and interactive CDs to instructor-led hands-on courses at locations around the world. Visit the Customer Education section of `ni.com` for online course schedules, syllabi, training centers, and class registration.

System Integration

If you have time constraints, limited in-house technical resources, or other dilemmas, you may prefer to employ consulting or system integration services. You can rely on the expertise available through our worldwide network of Alliance Program members. To find out more about our Alliance system integration solutions, visit the System Integration section of `ni.com`.

Worldwide Support

National Instruments has offices located around the world to help address your support needs. You can access our branch office Web sites from the Worldwide Offices section of `ni.com`. Branch office Web sites provide up-to-date contact information, support phone numbers, e-mail addresses, and current events.

If you have searched the technical support resources on our Web site and still cannot find the answers you need, contact your local office or National Instruments corporate. Phone numbers for our worldwide offices are listed at the front of this manual.

Glossary

Prefix	Meaning	Value
n-	nano-	10^{-9}
µ-	micro-	10^{-6}
m-	milli-	10^{-3}

A

active window — The window that user input affects at a given moment. The title of an active window is highlighted.

Array Display — A mechanism for viewing and editing numeric arrays.

auto-exclusion — A mechanism that prevents pre-existing lines from executing in the Interactive Execution Window.

B

binary control — A function panel control that resembles a physical on/off switch and can produce one of two values depending on the position of the switch.

bps — Bits per second.

breakpoint — An interruption in the execution of a program.

Breakpoint command — A specific command that interrupts the execution of a program.

button — A dialog box item that when selected executes a command associated with the dialog box.

C

checkbox	A dialog box item that allows you to toggle between two possible options.
Clipboard	A temporary storage area LabWindows/CVI uses to hold text that is cut, copied, or deleted from a work area.
CodeBuilder	The LabWindows/CVI feature that creates code based on a `.uir` file to connect your GUI to the rest of your program. This code can be compiled and run as soon as it is created.
common control	A control on a Common Control Function Panel that specifies a parameter in all functions associated with a Function Panel window.
compiler define	A command-line argument passed to the compiler that defines an identifier as a macro to the preprocessor.
control	An input and output device that appears on a function panel for specifying function parameters and displaying function results.
cursor	The flashing rectangle that shows where you can enter text on the screen. If you have a mouse installed, there is also a mouse cursor.
cursor location indicator	An element of the LabWindows/CVI screen that specifies the row and column position of the cursor in the window.

D

default command	The action that takes place when <Enter> is pressed and no command is specifically selected. Default command buttons are indicated in dialog boxes with an outline.
dialog box	A prompt mechanism in which you specify additional information needed to complete a command.

E

entry mode indicator	An element of the LabWindows/CVI screen that indicates the current text mode as either insert or overwrite.
excluded code	Code that is ignored during compilation and execution. Excluded lines of code are displayed in a different color than included lines of code.

F

`.fp` file	A file containing information about the function tree and function panels of an instrument module.
function panel	A screen-oriented user interface to the LabWindows/CVI libraries in which you can interactively execute library functions and generate code for inclusion in a program.
Function Panel Editor window	The window in which you build a function panel.
Function Panel window	The window that contains function panels.
function tree	The hierarchical structure in which the functions in a library or an instrument driver are grouped. The function tree simplifies access to a library or instrument driver by presenting functions organized according to the operation they perform, as opposed to a single linear listing of all available functions.
Function Tree Editor window	The window in which you build the skeleton of a function panel file.

G

generated code box	A text box located at the bottom of the function panel window that displays the function call that corresponds to the current state of the function panel controls.

global control	A function panel control that displays the contents of global variables in a library function. Global controls allow you to monitor global variables in a function that the function does not specifically return as results. These are read-only controls that the user cannot alter and do not contribute a parameter to the generated code.

H

hex	Hexadecimal.
highlight	The way in which input focus is displayed on a LabWindows/CVI screen; to move the input focus onto an item.

I

immediate action menu	A menu that has no menu items associated with it and causes a command to execute immediately. An immediate action command is suffixed with an exclamation point (!).
input control	A function panel control that accepts a value typed in from the keyboard. An input control can have a default value associated with it. This value appears in the control when the panel is first displayed.
input focus	Displayed on the screen as a highlight on an item, signifying that the item is active. User input affects the item in the dialog box that has the input focus.
instrument driver	A set of high-level functions for controlling an instrument. It encapsulates many low-level operations, such as data formatting and GPIB, RS-232, and VXI communication, into intuitive, high-level functions. An instrument driver can pertain to one particular instrument or to a group of related instruments. An instrument driver consists of a program and a set of function panels. The program contains the code for the high-level functions. Associated with the instrument program is an include file that declares the high-level functions you can call, the global variables you can access, and the defined constants you can use.
Interactive Execution window	A LabWindows/CVI work area in which sections of code may be executed without creating an entire program.

L

list box	A dialog box item that displays a list of possible choices.

M

MB	Megabytes of memory.
menu	An area accessible from a menu bar that displays selectable menu items.

N

new style (function definition)	A function definition in which parameters are declared directly in the parameter list.

O

old style (function definition)	A function definition in which parameters are declared outside of the parameter list.
output control	A function panel control that displays a value that the function you execute generates. An output control parameter must be a string, an array, or a reference parameter of type integer, long, single-precision, or double-precision.

P

Project window	A window containing a list of files your application uses.
prompt command	A command that requires additional information before it can be executed; a prompt command appears on a pull-down menu suffixed with three ellipses (…).

R

return value control	A function panel control that displays a value returned from a function as a return value rather than as a formal parameter.
ring control	A function panel control that represents a range of values much like the slide control but displays only a single item in a list rather than displaying the whole list at once as the slide control does. Each item has a different value associated with it. This value is placed in the function call.

S

scroll bars	Areas along the bottom and right sides of a window that show your relative position in the file. Scroll bars can be used with a mouse to move about in the window.
scrollable text box	A dialog box item that displays text in a scrollable display.
select	To choose the item that the next executed action will affect by moving the input focus (highlight) to a particular item or area.
shortcut key commands	A combination of keystrokes that provide a means of executing a command without accessing a menu in the menu bar.
slide control	A function panel control that resembles a physical slide switch. A slide control is a means for selecting one item from a list of options; it inserts a value in a function call that depends on the position of the crossbar on the switch.
slider	The crossbar on the slide control that determines the value placed in the function call.
Source window	A LabWindows/CVI work area in which programs are edited and executed.
Standard Input/Output window	A LabWindows/CVI work area in which textual output to and input from the user take place.
standard libraries	The LabWindows/CVI User Interface, Analysis, Data Formatting and I/O, GPIB, GPIB-488.2, DDE, TCP, RS-232, Utility, and C system libraries.

String Display window — A window for viewing and editing string variables and arrays.

T

text box — A dialog box item in which text is entered from the keyboard.

U

User Interface Editor window — The window in which you build pull-down menus, dialog boxes, panels, and controls and save them to a User Interface Resource (`.uir`) file.

V

Variables window — A window that shows the values of all the currently active variables.

W

Watch window — A window that shows the values of user-selectable variables and expressions that are currently active.

window — A working area that supports specific tasks related to developing and executing programs.

Index

A

About LabWindows/CVI command, Help menu, 3-76
ActiveX Container Support option, Create Distribution Kit dialog box, 3-27
ActiveX controllers and servers. *See* Tools menu.
Add File to Project command, File menu
    Source and Interactive Execution windows, 5-7
    User Interface Editor window, 4-4
Add File to Source Control command, Source Code Control submenu, 3-54
Add Files to Executable button, Target Settings dialog box, 3-14
Add .FP File to Project command, File menu
    adding help information, 9-6
    Function Panel Editor, 8-3
    function panel windows, 6-8
    Function Tree Editor, 7-3
Add Missing Includes command, Build menu, 5-21
Add Program File to Project command, File menu
    adding help information, 9-6
    Function Panel Editor, 8-3
    Function Panel windows, 6-7
    Function Tree Editor, 7-3
Add Watch Expression command
    Code menu, 6-7, 6-14
    Options menu, 10-2, 10-14
    Run menu, 5-27
Add/Edit Tools Menu Item dialog box, 3-73
Add/Edit Watch Expression command, Edit menu, 10-5 to 10-6
Add/Edit Watch Expression dialog box, 10-2
Align command
    Arrange menu, 4-13
    Edit menu, 8-5
Align Horizontal Centers command, Arrange menu, 4-13
Alignment command
    Arrange menu, 4-13
    Edit menu, 8-5
All Callbacks command, Generate menu, 4-18
All Code command, Generate submenu, 4-16 to 4-17
All Files command, Add Files to Project dialog box, 3-7
Alphabetize option, Select Function Panel dialog box, 3-51
Always Append Code to End option, Preferences command, 4-20
ANSI C Library display, External Compiler support dialog box, 3-22
Any Array data type, 3-43
Any Type data type, 3-43 to 3-44
Application File option, Target Settings dialog box, 3-12
Application Icon File option, Target Settings dialog box, 3-12
Application Title option, Target Settings dialog box, 3-12
applications, creating, 2-4 to 2-5
Apply Default Font command, Edit menu, 4-10
Arrange menu, User Interface Editor
    Align command, 4-13
    Align Horizontal Centers command, 4-13
    Alignment command, 4-13
    Center Label command, 4-15
    Control Coordinates command, 4-15
    Control ZPlane Order command, 4-15
    Distribute command, 4-14
    Distribution command, 4-14

Array data types, user-defined, 3-45
Array Display command, View menu, 10-8, 11-1
Array Display window
  Edit menu, 11-5
  File menu, 11-4
  Format menu, 11-7
  Help menu, 11-8
  invoking, 11-1
  Options menu, 11-8
  purpose and use, 2-4, 11-1 to 11-2
  Run menu, 11-7
  single-dimensional array (figure), 11-2
  Window menu, 11-7
arrays
  multi-dimensional arrays
    illustration, 11-2
    Reset Indices dialog box, 11-2 to 11-3
    specifying dimensions, 11-3
  one-dimensional array
    displaying in Array Display window (figure), 11-2
    Graphical Array view (figures), 10-9 to 10-10
  two-dimensional array, Graphical Array view (figure), 10-11 to 10-12
ASCII text format
  loading objects into User Interface Editor window, 4-23
  saving contents of User Interface Editor window in, 4-23
Assign Missing Constants command, Options menu, 4-23
Attach and Edit Source command, Edit Instrument dialog box, 3-51, 7-8
attribute constants, selecting, 6-10 to 6-11
attribute values, selecting, 6-11
Attributes for Child Panels section, Edit Panel dialog box, 4-8
Auto Save Project command, File menu, 3-5

B

background color preference, User Interface Editor windows, 4-22
Balance command, Edit menu, 5-10
Batch Build command, Build menu, 3-11
Beginning/End of Selection command, View menu, 5-15
bin directory (table), 1-3
Binary command, Create menu, 8-8 to 8-9
binary control parameters, specifying, 6-6
Bottom Edges option
  Alignment command, 4-13
  Distribution command, 4-14
Bracket Styles command, Options menu, 5-32
brackets
  finding pairs of, 5-10
  setting location for, 5-32
Break at First Statement command, Run menu
  Project window, 3-34
  Source and Interactive Execution windows, 5-23, 5-26
break key, enabling global Ctrl+F12 debug break key, 3-69
Break On First Chance Exceptions option, Run Options command, 3-68
Break on Library Errors option, Run Options command, 3-68
breakpoints. *See also* watch variables/expressions.
  applicable only in source code modules (note), 5-22
  breakpoint state, 5-22 to 5-23
  conditional, 5-23
  Edit Breakpoint dialog box, 5-26
  purpose and use, 5-22 to 5-23
  resuming execution, 5-23
  setting and clearing, 5-23
Breakpoints command, Run menu
  opening Breakpoints dialog box, 5-26
  options, 3-34 to 3-35
  setting breakpoints, 5-23

Breakpoints dialog box, 5-26 to 5-27
Add/Edit Item button, 5-26
buttons, 5-26 to 5-27
Edit Breakpoint dialog box, 5-26
options and buttons, 3-34 to 3-35
Bring Panel to Front command, View menu, 4-12
Browse Info window, using only one, 3-70
Browser. *See* Source Code Browser, Tools menu.
build errors
Build Errors command, Window menu, 3-60
Build Errors in Next File command, View menu, 5-21
Next Build Error command, View menu, 5-21
Previous Build Error command, View menu, 5-21
Show Build Error window for warnings option, 3-66
Build Information section, Create Distribution Kit dialog box, 3-23 to 3-25
Build menu
Project window, 3-8 to 3-30
Batch Build command, 3-11
Compile File command, 3-20
Configuration command, 3-8 to 3-9
Create Debuggable Dynamic Link Library command, 3-9
Create Debuggable Executable command, 3-9
Create Distribution Kit command, 3-23 to 3-30
Create Release Dynamic Link Library command, 3-10
Create Static Library command, 3-10
External Compiler Support command, 3-21 to 3-22
Mark All for Compilation command, 3-21
Mark File for Compilation command, 3-20
Target Settings command, 3-12 to 3-20
Target Type command, 3-11
Source and Interactive Execution windows
Add Missing Includes command, 5-21
Clear Interactive Declarations command, 5-3, 5-21
Compile File command, 5-18 to 5-19
Create Debuggable Dynamic Link Library command, 5-19
Create Debuggable Executable command, 5-19
Create Release Dynamic Link Library command, 5-20
Create Release Executable command, 5-19 to 5-20
Create Static Library command, 5-20
Generate Prototypes command, 5-21
Insert Include Statements command, 5-21
Mark File for Compilation command, 5-20
Next Build Error command, 5-21
Previous Build Error command, 5-21
Build Options command, Options menu, 3-62 to 3-66
Button Bar option, Find command
Source and Interactive Execution windows, 5-13
Variables window, 10-4
buttons, adding and positioning on toolbar, 5-2

C

Callback Function option
Edit Control dialog box, 4-8
Edit Menu Bar dialog box, 4-6
Edit Panel dialog box, 4-7

callback functions
associated with close controls (note), 4-16
generating code for
All Callbacks command, 4-18
Control Callbacks command, 4-18 to 4-19
Main Function command, 4-17 to 4-18
Menu Callbacks command, 4-19
Panel Callbacks command, 4-18
calling convention, default, 3-62
Cascade Windows command, Window menu, 3-59
Case Sensitive option, Find command
Source and Interactive Execution windows, 5-11
Variables window, 10-4
Case Sensitive option, Find UIR Objects dialog box, 4-11
Center Label command, Arrange menu, 4-15
Change Control Type command, Edit menu, 8-5
Change Control Type option, Function Panel Editor menu bar, 8-2
Change Format command, Options menu, 6-19
Character Select mode, 5-5
Check Foreground Lockout Settings on Startup, Environment dialog box, 3-70
Check In command, Source Code Control submenu, 3-54
Check Out command, Source Code Control submenu, 3-54
Checked option, Edit Menu Bar dialog box, 4-7
Child Panels Attributes section, Edit Panel dialog box, 4-8
child structure, 10-2
Class command, Create menu, 7-6
Clear Interactive Declarations command
Build menu, 5-3, 5-21
Code menu, 6-9
Clear Source Code Control Error Window command, Source Code Control submenu, 3-55
Clear Tags command, View menu, 5-16
Clear Window command, Edit menu, 5-3, 5-9
Close command, File menu
Array and String Display windows, 11-4
creating help information, 9-5
Function Panel Editor, 8-3
Function Panel windows, 6-7
Function Tree Editor, 7-2
Source and Interactive Execution windows, 5-7
User Interface Editor, 4-3
Close All command, Window menu, 3-60
Close Variable command, View menu, 10-2, 10-7
code. *See* source files.
Code menu
Function Panel windows, 6-8 to 6-14
Add Watch Expression command, 6-7, 6-14
Clear Interactive Declarations command, 6-9
Declare Variable command, 6-4, 6-5, 6-8 to 6-9
Insert Function Call command, 6-14
Run Function Panel command, 6-8
Select Attribute Constant command, 6-10 to 6-11
Select Attribute Value command, 6-11
Select UIR Constant command, 6-9 to 6-10
Set Target File command, 6-14
View Variable Value command, 6-7, 6-14, 11-1, 11-3

User Interface Editor
Generate submenu, 4-15 to 4-19
All Callbacks command, 4-18
All Code command, 4-16 to 4-17
Control Callbacks command, 4-18 to 4-19
Generate All Code dialog box, 4-16
Main Function command, 4-17 to 4-18
Menu Callbacks command, 4-19
Panel Callback command, 4-18
Preferences command
Always Append Code to End option, 4-20
Default Control Events option, 4-20
Default Panel Events option, 4-20
Set Target File command, 4-15
View command, 4-19
code modules
adding to projects, 3-6
listing in Project window, 2-4
CodeBuilder overview, 4-2 to 4-3.
See also Generate menu.
color coding tokens in source and include files, 5-32 to 5-33
Coloring tool, 4-1
colors, setting in Editor Preferences dialog box, 4-22
Colors command, Options menu
Project window, 3-75
Source and Interactive Execution windows, 5-32
Column Select mode, 5-6
Command Line command, Options menu, 3-69
Common Control function panel, 6-6
Common Control Panel command, Create menu, 8-13
comparing source files. *See* Diff command, Edit menu.
Compatibility With option, 3-62
compile errors, maximum number of, 3-66
Compile File command, Build menu
Project window, 3-20
Source and Interactive Execution windows, 5-18 to 5-19
compiled files, including in project, 2-4
compiler defines
predefined macros, 3-67
syntax, 3-66
Compiler Defines command, Options menu, 3-66 to 3-67
compiler options
Compatibility with, 3-62
Debugging level, 3-63
Default calling convention, 3-62
Detect assignments in conditional expressions, 3-65
Detect signed/unsigned pointer mismatches, 3-64
Detect unitialized local variables at run-time, 3-63
Detect unreachable code, 3-64 to 3-65
Detect unreferenced identifiers, 3-65
Display status dialog during build, 3-64
Generate source code browse information, 3-66
Image base address, 3-63
Make 'O' option compatible with CVI 5.0.1, 3-64
Maximum number of compile errors, 3-66
Maximum stack size, 3-62
Prompt for include file paths, 3-63
Require Function Prototypes, 3-65, 5-4
Require function prototypes, 3-65
Require return values for non-void functions, 3-65 to 3-66

Show Build Error window for warnings, 3-66
Stop on first file with errors, 3-66
Track include file dependencies, 3-63
Unitialized local variables detection, 3-64
compiler support, external. *See* External Compiler Support dialog box.
compiling files. *See* Build menu.
conditional breakpoints, 5-23
conditional expressions, detecting assignments in, 3-65
Configuration command submenu, Build menu, 3-8 to 3-10
Create Debuggable Dynamic Link Library command, 3-9
Create Debuggable Executable command, 3-9
Create Release Dynamic Link Library command, 3-10
Create Release Executable command, 3-10
Create Static Library command, 3-10
Debug option, 3-8
Release option, 3-9
configuring LabWindows/CVI
date and time options (DSTRules), 1-4
directory options, 1-3
cvidir, 1-3
tmpdir, 1-3
font options, 1-4 to 1-5
DialogFontBold, 1-4
DialogFontName, 1-4
DialogFontSize, 1-4
MenuFontBold, 1-5
MenuFontName, 1-5
MenuFontSize, 1-5
setting, 1-2
startup options (table), 1-1
string value for Registry (figure), 1-2
timer options (useDefaultTimer), 1-4
console application, creating, 3-14
Console window, using for standard I/O when debugging, 3-70
Constant Name option
Edit Control dialog box, 4-8
Edit Menu Bar dialog box, 4-6
Edit Panel dialog box, 4-7
constants
assigning names, 4-23
selecting user interface constants, 6-9 to 6-10
Contents command, Help menu, 3-75
context menu
Function Tree Editor, 7-1 to 7-2
Source window, 5-3
Continue command, Run menu
Project window, 3-34
Source and Interactive Execution windows, 5-25
Control command
Edit menu, 4-8 to 4-9
Help menu, 6-19
Control Appearance section, Edit Label/Value Pairs dialog box, 4-9
Control Callbacks command, Generate menu, 4-18 to 4-19
Control Coordinates command
Arrange menu, 4-15
Edit menu, 8-5 to 8-6
Control Help command, Edit menu, 8-6
Control Help option, Function Panel Editor menu bar, 8-2
Control Settings section, Edit Control dialog box, 4-9
Control Style command, Edit menu, 4-10
Control ZPlane Order command, Arrange menu, 4-15
controls. *See also* function panel controls.
changing control type (example), 8-23 to 8-24
cutting and pasting (example), 8-24 to 8-25

help information, 9-4
preferences, 4-23
Controls command, Create menu, 4-10
conventions used in manual, *xxiii-xxiv*
Convert UI to Lab Style command, Tools menu, 3-59
Copy command, Edit menu
adding help information, 9-6
Function Tree Editor, 7-3
Source and Interactive Execution windows, 5-8
User Interface Editor window, 4-5
Copy Controls command, Edit menu, 8-4
Copy Controls option, Function Panel Editor menu bar, 8-2
Copy Panel command, Edit menu
Function Panel Editor, 8-4
User Interface Editor window, 4-5
Create ActiveX Controller command, Tools menu
Function Panel Editor, 8-16
Function Tree Editor, 7-9
Project window, 3-52
Source and Interactive Execution windows, 5-29
Create ActiveX Server command, Tools menu
Function Panel Editor, 8-16
Function Tree Editor, 7-9
Project window, 3-53
Source and Interactive Execution windows, 5-29
Create Binary Control dialog box, 8-8
Create Console Application, Target Settings dialog box, 3-14
Create Debuggable Dynamic Link Library command
Build menu, 5-19
Configuration submenu, 3-9
Create Debuggable Executable command
Build menu, 5-19
Configuration submenu, 3-9
Create Distribution Kit command, Build menu, 3-23
Create Distribution Kit dialog box, 3-23 to 3-30
Advanced button, 3-30
Build button, 3-30
Build Information section, 3-23 to 3-25
Browse option, 3-24 to 3-25
Build Location option, 3-23
Install ActiveX Container Support option, 3-27
Install DataSocket Support control, 3-26
Install Low-Level Support Driver option, 3-27
Install NI Reports Support control, 3-26 to 3-27
Install Win95 DCOM/RTE Support option, 3-27
Installation Language option, 3-23 to 3-24
Run-Time Engine Install Location option, 3-26
Run-Time Engine Support, 3-24 to 3-26
Cancel button, 3-30
Default button, 3-30
File Groups section, 3-28 to 3-30
Add Group option, 3-28
Create Shortcuts option, 3-28 to 3-29
Distribute Objects/Libraries For Both Compilers option, 3-29 to 3-30
Edit Group option, 3-28
File Groups option, 3-28
Register Files As ActiveX Servers option, 3-30
Install Location option, 3-23
Create DLL Project command, Options menu, 7-13
Create Global Variable Control dialog box, 8-12
Create Input Control dialog box, 8-7

Create IVI Instrument Driver command, Tools menu
Function Panel Editor, 8-17
Function Tree Editor, 7-9
Project window, 3-53
Source and Interactive Execution windows, 5-29
Create menu
Function Panel Editor, 8-7 to 8-13
Binary command, 8-8 to 8-9
Common Control Panel command, 8-13
Function Panel command, 8-12
Global Variable command, 8-12
Input command, 8-7
Message command, 8-12
Numeric command, 8-10 to 8-11
Output command, 8-11
Return Value command, 8-11 to 8-12
Ring command, 8-9
Slide command, 8-8
Function Tree Editor, 7-5 to 7-7
Class command, 7-6
Function Panel Window command, 7-6 to 7-7
Instrument command, 7-5
User Interface Editor
Controls command, 4-10
Menu Bar command, 4-10
Panel command, 4-10
Create Numeric Control dialog box, 8-10 to 8-11
Create Object File command, Options menu, 5-34
Create Output Control dialog box, 8-11
Create Release Dynamic Link Library command
Build menu, 5-20
Configuration submenu, 3-10
Create Release Executable command
Build menu, 5-19 to 5-20
Configuration submenu, 3-10
Create Return Value Control dialog box, 8-11 to 8-12
Create Ring Control dialog box, 8-9
Create Slide Control dialog box, 8-8
Create Static Library command, Build menu
Project window, 3-10
Source and Interactive Execution windows, 5-20
curly braces
finding pairs of, 5-10
setting location for, 5-32
Current Tree command, View menu
Function Panel Editor, 8-14
Function Panel windows, 6-15
customer education, B-1
Cut command, Edit menu
adding help information, 9-6
Function Tree Editor, 7-3
Source and Interactive Execution windows, 5-8
User Interface Editor window, 4-5
Cut Controls command, Edit menu, 8-4
Cut Controls option, Function Panel Editor menu bar, 8-2
Cut Panel command, Edit menu
Function Panel Editor, 8-4
User Interface Editor window, 4-5
CVI Environment sleep Policy option, 3-69 to 3-70
CVI Libraries display, External Compiler support dialog box, 3-22
CVI macro, 3-67
_CVI_DEBUG macro, 3-67
_CVI_DLL_ macro, 3-67
_CVI_EXE_ macro, 3-67
_CVI_LIB_ macro, 3-67
cvidir configuration option, 1-3

D

data tool tips, enabling, 3-70
data type compatibility for function panel variables, 6-13
Data Types command, Options menu, 8-18
data types for instrument drivers, 3-41 to 3-47
  browsing, 3-56
  overview, 3-41 to 3-42
  predefined, 3-42 to 3-44
    intrinsic C data types, 3-42 to 3-43
    meta data types, 3-43 to 3-44
  user-defined, 3-44 to 3-45
    array data types, 3-45
    creating, 3-45
  VISA data types, 3-46
DataSocket Support control, Create Distribution Kit dialog box, 3-26
date option, DSTRules, 1-4
dates
  Show Full Dates command, View menu, 3-8
  Sort by Date command, View menu, 3-8
daylight savings time, setting, 1-4
Debug Output window
  bringing to front whenever modified, 3-70
  Debug Output command, 3-60
  setting number of lines to display, 3-70
debugging
  Create Debuggable Dynamic Link Library command, 3-9, 5-19
  Create Debuggable Executable command, 3-9, 5-19
  Debug command, Run menu, 3-31
  Debug option, Configuration command, 3-8
  Debug/Run Interactive Statements command, Run menu, 5-24 to 5-25
  DLLs, 3-31 to 3-35
    location of required files, 3-32 to 3-33
    running external process, 3-33 to 3-34
    running program in LabWindows/CVI, 3-33
  Use Console Window for Standard I/O When Debugging option, 3-70
debugging levels
  Extended, 3-63
  No Run-time Checking, 3-63
  Standard, 3-63
decimal symbol, localized, 4-22
Declare Variable command, Code menu
  defining variables, 6-4
  Function Panel windows, 6-8 to 6-9
  specifying output control parameter, 6-6
Declare Variable dialog box, 6-8 to 6-9
  Add declaration to current block in target file checkbox, 6-8
  Add declaration to top of target file checkbox, 6-8
  Cancel button, 6-9
  Execute declaration, 6-8
  Number of Elements, 6-8
  OK button, 6-9
  Set Target File button, 6-8
  Variable Name, 6-8
  Variable Type, 6-8
DEF file, generating, 7-12
__DEFALIGN macro, 3-67
Default All command, Options menu, 6-18
Default calling convention option, 3-62
Default Control command, Options menu, 6-18
Default Control Events option, Preferences command, 4-20
Default Panel Events option, Preferences command, 4-20
Delete command, Edit menu
  adding help information, 9-6

Source and Interactive Execution windows, 5-9
User Interface Editor window, 4-5
Delete Watch Expression command, Edit menu, 10-6
Detach Program command, Edit Instrument dialog box, 3-51, 7-8
Detect assignments in conditional expressions option, 3-65
Detect signed/unsigned pointer mismatches option, 3-64
Detect unreachable code option, 3-64 to 3-65
Detect unreferenced identifiers option, 3-65
DialogFontBold option, 1-4
DialogFontName option, 1-4
DialogFontSize option, 1-4
Diff command, Edit menu, 5-10
Diff With, 5-10
Find Next Difference, 5-10
Ignore White Space, 5-10
Match Criteria, 5-10
Recompare Ignoring White Space, 5-10
Synchronize at Top, 5-10
Synchronize Selections, 5-10
Dimmed option, Edit Menu Bar dialog box, 4-7
directory configuration options, 1-3
Display Entire Buffer command, Options menu, 11-8
Display status dialog during build option, 3-64
Distribute command, Edit menu, 8-5
distributing standalone executables. *See* standalone executables, creating and distributing.
Distribution command
Arrange menu, 4-14
Edit menu, 8-5
Distribution Kit. *See* Create Distribution Kit dialog box.
DLL File option, Target Settings dialog box, 3-15
DLLMain function, generating, 4-18
DLLs
adding to project, 3-6
Create Debuggable Dynamic Link Library command, 3-9, 5-19
Create DLL Project command, Options menu, 7-13
Create Release Dynamic Link Library command, 3-9, 5-20
debugging, 3-31 to 3-35
location of required files, 3-32 to 3-33
running external process, 3-33 to 3-34
running program in LabWindows/CVI, 3-33
effects of VXIplug&play Style command, 7-13 to 7-14
generating DLL import library, 5-34
generating source code for DLL import library, 5-33
Target Settings dialog box, 3-15 to 8-20
Where to Copy DLL option, 3-15
documentation
conventions used in manual, *xxiii-xxiv*
LabWindows/CVI documentation set, *xxiv-xxv*
documenting instrument drivers, 3-50
Down Call Stack command, Run menu, 5-27
DSTRules option, 1-4

E

Edit ActiveX Server command, Tools menu
Function Panel Editor, 8-17
Function Tree Editor, 7-9
Program window, 3-53
Source and Interactive Execution windows, 5-29
Edit Breakpoint dialog box, 5-26
Edit Character command, Edit menu, 11-5

Edit command, Instrument menu. *See also* Edit Instrument dialog box.
Function Panel Editor, 8-16
Function Tree Editor, 7-1, 7-8
Project window, 3-51
Edit Control command, Edit menu, 8-4
Edit Control dialog box
Control Appearance section, 4-9
Control Settings section, 4-9
Edit Label/Value Pairs dialog box, 4-9
Label Appearance section, 4-9
Quick Edit Window, 4-9
Source Code Connection, 4-8
Edit Control option, Function Panel Editor menu bar, 8-2
Edit Custom Controls command, Edit menu, 4-10
Edit Custom Controls dialog box, 4-10
Edit Data type List dialog box, 8-18
Edit Function command, Edit menu, 8-5
Edit Function Panel command, Tools menu
adding help information, 9-7
Source and Interactive Execution windows, 5-30
Edit Function Panel Window command
Edit menu, 7-4
Options menu, 6-19
Edit Function Panel Window option, Function Tree Editor context menu, 7-1
Edit Function Tree command
Edit Instrument dialog box, 3-51, 7-8
Options menu, 8-19
Tools menu, 5-30, 9-7
Edit Help command, Edit menu, 7-3
Edit Instrument Attributes command, Tool menu
Function Panel Editor, 8-17
Function Tree Editor, 7-9
Source and Interactive Execution windows, 5-30
Edit Instrument dialog box
Function Panel Editor, 8-16
Project window, 3-51
Edit Label/Value Pairs dialog box, 4-9
Edit menu
Array Display window
Edit Value command, 11-5
Find command, 11-5
Goto command, 11-5
creating help information, 9-6 to 9-7
Function Panel Editor, 8-3 to 8-7
Align command, 8-5
Alignment command, 8-5
Change Control Type command, 8-5
Control Coordinates command, 8-5 to 8-6
Control Help command, 8-6
Copy Controls command, 8-4
Copy Panel command, 8-4
Cut Controls command, 8-4
Cut Panel command, 8-4
Distribute command, 8-5
Distribution command, 8-5
Edit Control command, 8-4
Edit Function command, 8-5
Find command, 8-6
Function Help command, 8-7
Paste command, 8-4
Redo command, 8-3 to 8-4
Replace command, 8-6
Undo command, 8-3 to 8-4
Window Help command, 8-7
Function Tree Editor, 7-3 to 7-5
Copy command, 7-3
Cut command, 7-3
Edit Function Panel Window command, 7-4
Edit Help command, 7-3
Edit Node command, 7-3
Find command, 7-5

.FP Auto-Load List command, 7-4
Paste Above command, 7-3
Paste Below command, 7-3
Replace command, 7-5
Project window, 3-6 to 3-7
Add Files to Project command, 3-6 to 3-7
Exclude File from Build command, 3-7
Include File in Build command, 3-7
Move Item Down command, 3-7
Move Item Up command, 3-7
Remove File command, 3-7
Replace File in Project command, 3-7
Select All command, 3-7
Workspace command, 3-6
Source and Interactive Execution windows
Balance command, 5-10
Clear Window command, 5-3, 5-9
Copy command, 5-8
Cut command, 5-8
Delete command, 5-9
Diff command, 5-10
Find command, 5-11 to 5-13
Go To Definition command, 5-10
Insert Construct command, 5-9 to 5-10
Next File command, 5-14
Paste command, 5-8
Redo command, 5-8
Replace command, 5-14
Resolve All Excluded Lines command, 5-9
Select All command, 5-9
Toggle Exclusion command, 5-3, 5-9
Undo command, 5-8
String Display window
Edit Character command, 11-5
Edit Mode command, 11-5
Find command, 11-6
Goto command, 11-6
Overwrite command, 11-6
User Interface Editor
Apply Default Font command, 4-10
Control command, 4-8 to 4-9
Control Style command, 4-10
Copy command, 4-5
Copy Panel command, 4-5
Cut command, 4-5
Cut Panel command, 4-5
Delete command, 4-5
Edit Custom Controls command, 4-10
Menu Bars command, 4-6 to 4-7
Panel command, 4-7 to 4-8
Paste command, 4-5
Redo command, 4-4
Set Default Font command, 4-10
Tab Order command, 4-9
Undo command, 4-4
when commands are enabled (note), 4-4
Variables window
Edit Value command, 10-3 to 10-4
Find command, 10-4 to 10-5
Next Scope command, 10-5
Previous Scope command, 10-5
Watch window
Add/Edit Watch Expression command, 10-5 to 10-6
Delete Watch Expression command, 10-6
Edit Value command, 10-5
Find command, 10-6
Edit Menu Bar dialog box, 4-6 to 4-7
Edit Mode command, Edit menu, 11-5
Edit Node command, Edit menu, 7-3
Edit Node option, Function Tree Editor context menu, 7-1
Edit Panel dialog box
Attributes for Child Panels section, 4-8

Panel Settings section, 4-7 to 4-8
Quick Edit Window section, 4-8
Source Code Connection section, 4-7
Edit Tabbing Order dialog box, 4-9
Edit Value command, Edit menu
Array Display window, 11-5
Variables window, 10-3 to 10-4
Watch window, 10-5
Edit Watch Expression command, Edit menu, 10-5
Edit Workspace dialog box, 3-6
Editing tool, 4-1
Editor Preferences command, Options menu, 5-31
Enable Auto Replace command, Tools menu
Function Panel Editor, 8-17
Function Tree Editor, 7-10
Enable global Ctrl+F12 debug break key, 3-69
End of Selection command, View menu, 5-15
Environment command, Options menu, 3-51, 3-69 to 3-71
environment options
Bring Debug Output window to front whenever modified, 3-70
Check Foreground Lockout Setting on Startup, 3-70
CVI Environment Sleep Policy, 3-69 to 3-70
Enable Data Tool Tips, 3-70
Force Loaded Instrument Driver's Source into Interactive window, 3-70
Force Project Compiled Source Files into Interactive window, 3-71
Goto source After Inserting Code from Function Panel, 3-70
Interactive Window Memory Size, 3-70
Lines in Debug Output window, 3-70
Use Console Window for Standard I/O When Debugging, 3-70
Use Only One Browse Info window, 3-70
Use Only One function panel window, 3-70
Error command, View menu
Function Panel Editor, 8-13
Function Panel windows, 6-15
errors
Break on library errors option, 3-68
build errors, 5-21
Display status dialog box during build option, 3-64
Maximum number of compile errors option, 3-66
run-time error reporting, 3-31, 5-25
Show Build Error window for warnings option, 3-66
Source Code Control Errors window, 3-61
Stop on first file with errors option, 3-66
Estimate Number of Elements command, Options menu, 10-14
Exclude File from Build command, Edit menu, 3-7
Exclude Function command, Options menu, 6-18
excluding lines of code, 5-9
executables, creating and distributing. *See* standalone executables, creating and distributing.
Execute command, Run menu, 3-35
Exit LabWindows/CVI command, File menu
Array and String Display windows, 11-4
creating help information, 9-5
Function Panel Editor, 8-2
Function Panel windows, 6-7
Function Tree Editor, 7-2
Project window, 3-5
Source and Interactive Execution windows, 5-6
User Interface Editor, 4-3
Variables and Watch windows, 10-3

Expand Variable command, View menu, 10-2, 10-7
Exports option, Target Settings dialog box, 3-19
expressions. *See also* watch variables/expressions.
  detecting assignments in conditional expressions option, 3-65
  regular expressions (table), 5-12 to 5-13
External Compiler Support command, 3-21
External Compiler Support dialog box, 3-21 to 3-22
  ANSI C Library, 3-22
  CVI Libraries, 3-22
  Other Symbols, 3-22
    Header File field, 3-22
    Object File field, 3-22
  UIR Callbacks Object File, 3-21
  Using LoadExternalModule to Load Object and Static Library Files, 3-21
external process
  debugging DLLs, 3-33 to 3-34
  selecting, 3-35
eyedropper tool, 4-2

F

file extensions, displaying project files in order of, 3-8
File Groups section, Create Distribution Kit dialog box, 3-28 to 3-30
File menu
  Array and String Display windows
    Close command, 11-4
    Exit LabWindows/CVI command, 11-4
    Input command, 11-4
    most recently closed files list, 11-4
    New command, 11-4
    Open command, 11-4
    Output command, 11-4
    Save command, 11-4
    Save All command, 11-4
creating help information, 9-5 to 9-6
Function Panel Editor, 8-2 to 8-3
  Add .FP File to Project command, 8-3
  Add Program File to Project command, 8-3
  Close command, 8-3
  Exit LabWindows/CVI command, 8-2
  New command, 8-2
  Open command, 8-2
  Read Only command, 8-3
  Save All command, 8-3
  Savecommand, 8-2
  Save Copy of .FP File command, 8-2
  Save .FP File command, 8-2
  Save .FP File As command, 8-2
Function Panel windows
  Add .FP File to Project command, 6-7
  Add Program File to Project command, 6-7
  Close command, 6-7
  Exit LabWindows/CVI command, 6-7
  most recently closed files list, 6-7
  New command, 6-7
  Open command, 6-7
  Save All command, 6-7
Function Tree Editor, 7-2
  Add .FP File To Project command, 7-3
  Add Program File To Project command, 7-3
  Close command, 7-2
  Exit LabWindows/CVI command, 7-2
  New command, 7-2
  Open command, 7-2

Read Only command, 7-3
Save All command, 7-3
Savecommand, 7-2
Save Copy of .FP File As command, 7-2
Save .FP File As command, 7-2
Save .FP File command, 7-2
Project window, 3-3 to 3-5
Auto Save Project command, 3-5
Exit LabWindows/CVI command, 3-5
most recently closed files list, 3-5
New command, 3-3
Open command, 3-4
Print command, 3-5
Save command, 3-4
Save All command, 3-5
Save Project As command, 3-4
Set Active Project command, 3-4
Source and Interactive Execution windows
Add File to Project command, 5-7
Close command, 5-7
Exit LabWindows/CVI command, 5-6
Hide command (note), 5-7
New command, 5-6
Open command, 5-6
Open Quoted Text command, 5-6
Print command, 5-7
Read Only command, 5-7
Save command, 5-6
Save All command, 5-7
Save As command, 5-6
Save Copy As command, 5-7
User Interface Editor
Add File to Project command, 4-4
Close command, 4-3
Exit LabWindows/CVI command, 4-3
New command, 4-3
Open command, 4-3
Print command, 4-4
Read Only command, 4-4
Save command, 4-3
Save All command, 4-3
Save As command, 4-3
Save Copy As command, 4-3
Variables and Watch windows
Exit LabWindows/CVI command, 10-3
Hide command, 10-3
most recently closed files list, 10-3
New command, 10-3
Open command, 10-3
Output command, 10-3
Save All command, 10-3
Savecommand, 10-3
<filename> startup option (table), 1-1
files. *See also* project files.
adding to project list, 3-6 to 3-7
browsing source code files, 3-55
format conversion when loading, 3-50
instrument driver files, 3-36 to 3-37
location of files required for debugging DLLs, 3-32 to 3-33
selecting multiple files in Project window, 3-2 to 3-3
Find command, Edit menu
Array Display window, 11-5
creating help information, 9-6
Function Panel Editor, 8-6
Function Tree Editor, 7-5
Source and Interactive Execution windows, 5-11 to 5-13
String Display window, 11-6
Variables window, 10-4 to 10-5
Watch window, 10-6
Find dialog box
Array Display window, 11-5
Source and Interactive Execution windows, 5-11 to 5-13

Variables window, 10-4 to 10-5
Find Function Panel command, View menu
Function Panel Editor, 8-14
Function Panel windows, 6-15 to 6-16
Source and Interactive Execution windows, 5-17 to 5-18
Find Next button, Find UIR Objects dialog box, 4-12
Find Next option, Find command
Source and Interactive Execution windows, 5-13
Variable Display and Watch Windows, 10-5
Find Prev button, Find UIR Objects dialog box, 4-12
Find Prev option, Find command
Source and Interactive Execution windows, 5-13
Variable Display and Watch Windows, 10-5
Find UIR Objects command, View menu, 4-11 to 4-12, 5-18
Find UIR Objects dialog box
Case Sensitive option, 4-11
Edit button, 4-12
Find button, 4-11
Find Next button, 4-12
Find option, 4-11
Find Prev button, 4-12
Regular Expression option, 4-11
search criteria in Search By ring control, 4-11
Stop button, 4-12
Whole Word option, 4-11
Wrap option, 4-11
Find What text box option, Find command, 5-11
Finish Function command, Run menu, 5-25
First Function Panel Window command, View menu
Function Panel Editor, 8-15
Function Panel windows, 6-16
First Panel command, View menu, 6-2
__FLAT__ macro, 3-67
Flatten option, Select Function Panel dialog box, 3-51, 6-2
Follow Pointer Chain command, View menu, 10-2, 10-7
Font command, Options menu
Project window, 3-74
Source and Interactive Execution windows, 5-32
font directory (table), 1-3
font options
DialogFontBold, 1-4
DialogFontName, 1-4
DialogFontSize, 1-4
MenuFontBold, 1-5
MenuFontName, 1-5
MenuFontSize, 1-5
fonts, setting and applying defaults, 4-10
Foreground lockout settings, checking on startup, 3-70
format conversion of files during loading, 3-50
Format menu
Array and String Display windows, 11-7
Variables and Watch windows, 10-12
.FP Auto-Load List command, Edit menu, 7-4
.FP File Format command, Options menu, 7-11
.fp files. *See* instrument driver function panel (.fp) files.
Full Runtime Engine option
Run-Time Engine Support, Create Distribution Kit dialog box, 3-24
Runtime Support option, Target Settings dialog box
Target Type Dynamic Link Library, 3-15
Target Type Executable, 3-12
function classes. *See* function trees.
Function command, Help menu, 6-19

Function Help command, Edit menu, 8-7
Function Panel command, Create menu, 8-12
function panel controls, 6-4 to 6-7
  binary control parameter, 6-4
  common control panel, 6-6
  global control, 6-6
  input control parameter, 6-5
  numeric control parameter, 6-5
  output control parameter, 6-6
  overriding with Toggle Control Style command, 6-18
  purpose and use, 6-4
  restoring default value, 6-18
  return value control parameter, 6-4
  slide control parameter, 6-5
  viewing arrays, structures, and variables, 6-7
Function Panel Editor, 8-1 to 8-25
  Create menu, 8-7 to 8-13
  Edit menu, 8-3 to 8-7
  examples, 8-20 to 8-25
    changing control type, 8-23 to 8-24
    creating Function window, 8-20 to 8-23
    cutting and pasting controls, 8-24 to 8-25
  File menu, 8-2 to 8-3
  Help menu, 8-19
  Instrument menu, 8-15 to 8-16
  invoking
    from function panel, 8-1
    from Function Tree Editor, 8-1
  menu bar, 8-1 to 8-2
  Options menu, 8-18 to 8-19
  Tools menu, 8-16 to 8-18
  View menu, 8-13 to 8-15
  Window menu, 8-18
Function Panel Editor windows
  adding help information (example), 9-9
  purpose and use, 2-4
Function Panel Help Editor window, 2-4
Function Panel History command, View menu, 6-15
Function Panel Tree command, View menu, 5-16
Function Panel Window command, Create menu, 7-6 to 7-7
Function Panel windows
  closing after executing Insert Function Call command, 3-70
  Code menu, 6-8 to 6-14
  File menu, 6-7
  generated code box, 6-3
  help information (old style), 9-4
  Help menu, 6-19
  Instrument menu, 6-17
  Library menu, 6-17
  multiple function panels per window, 6-3
  Options menu, 6-18 to 6-19
  purpose and use, 2-3
  Tools menu, 6-17
  using only one, 3-70
  View menu, 6-15 to 6-16
  Window menu, 6-17
function panels. *See also* function panel controls.
  accessing, 6-2 to 6-3
    from Instrument menu, 3-51 to 3-52
  building for instrument drivers, 3-48 to 3-49
  creating (example), 8-20 to 8-23
  cutting and pasting functions and panels (example), 7-15
  definition, 2-3, 6-1
  finding functions, 5-17 to 5-18
  help information (new style), 9-3 to 9-4
  invoking Function Panel Editor, 8-1
  multiple function panels per window, 6-3
  purpose and use, 6-1

recalling. *See* Recall Function Panel command, View menu.
using existing panels in new driver (example), 7-15 to 7-16
function prototypes, requiring, 3-65
Function subwindow, Variables window, 10-1
Function Tree Editor, 7-1 to 7-16
adding help information (example), 9-8
context menu, 7-1 to 7-2
Create menu, 7-5 to 7-7
Edit menu, 7-3 to 7-5
examples, 7-14 to 7-16
creating function tree with multiple classes, 7-14 to 7-15
cutting and pasting functions and panels, 7-15
editing items in function tree, 7-16
using existing function panels in new driver, 7-15 to 7-16
File menu, 7-2
Help menu, 7-14
Instrument menu, 7-7 to 7-8
invoking, 7-1
invoking Function Panel Editor, 8-1
Options menu, 7-11 to 7-14
Tools menu, 7-9 to 7-10
Window menu, 7-10
Function Tree Editor windows
opening
with New command, 3-3
with Open command, 3-4
purpose and use, 2-3
Function Tree Help Editor window, 2-4
function trees
adding or inserting functions, 7-7
building for instrument drivers, 3-48
classes
adding to empty tree or class, 7-6
help information, 9-3
inserting into existing tree, 7-6
overview, 6-2
creating, with multiple classes (example), 7-14 to 7-15
definition, 6-2, 7-1
editing items (example), 7-16
functions
adding to empty tree or class, 7-7
browsing, 3-56
cutting and pasting functions and panels (example), 7-15
inserting into existing tree, 7-7
requiring return values for non-void functions, 3-65 to 3-66
functions for instrument drivers, 3-40 to 3-41
building function panels, 3-48 to 3-49
building function tree, 3-48
defining
function parameters, 3-41
hierarchy of functions, 3-41
required functions, 3-47 to 3-48
structuring functions, 3-40 to 3-41
writing function code, 3-49

G

Generate All Code dialog box, 4-16
Generate DEF File command, Options menu, 7-12
Generate DLL Import Library command, Options menu, 5-34
Generate DLL Import Source command, Options menu, 5-33
Generate Documentation command, Options menu, 7-12
Generate Function Prototypes command, Options menu, 7-12
Generate IVI C++ Wrapper command, Tools menu, 7-9
Generate Main Function dialog box, 4-17
Generate menu, 4-15 to 4-19
All Callbacks command, 4-18
All Code command, 4-16 to 4-17

Control Callbacks command, 4-18 to 4-19
Generate All Code dialog box, 4-16
Main Function command, 4-17 to 4-18
Menu Callbacks command, 4-19
Panel Callback command, 4-18
Generate New Source For Function Tree command, Tools menu, 7-10
Generate ODL File command, Options menu, 7-12
Generate Prototypes command, Build menu, 5-21
Generate Source For Function Node option, Function Tree Editor context menu, 7-1
Generate Source For Function Panel command, Tools menu, Function Panel Editor, 8-17
Generate Visual Basic Include command, Options menu, 5-34
Generate Windows Help command, Options menu, 7-12
Generate WinMain() Instead of Main() checkbox, Generate Main Function dialog box, 4-18
generated code box, Function Panel window, 6-3
Get Latest Version command, Source Code Control submenu, 3-54
Get Latest Version of All command, Source Code Control submenu, 3-54
global control, 6-6
Global subwindow, Variables window, 10-1
Global Variable command, Create menu, 8-12
Go To Cursor command, Run menu, 5-25
Go To Declaration command, Tools menu
Function Panel Editor, 8-18
Function Tree Editor, 7-10
Go To Declaration option, Function Tree Editor context menu, 7-2
Go To Definition command
Edit menu, 5-10
Tools menu
Function Panel Editor, 8-17
Function Tree Editor, 7-10
View menu, 10-8
Go To Definition option, Function Tree Editor context menu, 7-2
Go To Execution Position command, View menu, 10-8
Goto command, Edit menu
Array Display window, 11-5
String Display window, 11-6
Graphical Array view, 10-9 to 10-12
1D arrays (figures), 10-9 to 10-10
2D arrays (figures), 10-11 to 10-12
Graphical Array View command, View menu
Array and String Display windows, 11-7
Variable and Watch windows, 10-9 to 10-12

H

header files
including in project, 3-7
optional in project file list, 3-1
previewing, 4-12
Help dialog box, for functions, windows, or classes, 3-52
help information, 9-1 to 9-11
control help, 9-4
Edit menu, 9-6 to 9-7
editing, 9-2
examples, 9-8 to 9-11
copying and pasting help text, 9-10 to 9-11
Function Panel Editor, 9-9
Function Tree Editor, 9-8
File menu, 9-5 to 9-6
function class help, 9-3
function help (new style help only), 9-3 to 9-4

function panel window help (old style help only), 9-4
Generate Windows Help command, Options menu, 7-12
Help menu, 9-7
help options, 9-2
Help Style command, Options menu, 7-11 to 7-12
instrument help, 9-3
new style versus old style help, 9-1
Tools menu, 9-7
Transfer Window Help to Function Help command, Options menu, 7-12
types of help information (table), 9-2
Window menu, 9-7
Help menu
Array and String Display windows, 11-8
creating help information, 9-7
Function Panel Editor, 8-19
Function Panel windows
Control command, 6-19
Function command, 6-19
Function Tree Editor, 7-14
Project window
About LabWindows/CVI command, 3-76
Contents command, 3-75
Tip of the Day, 3-76
Web Links, 3-76
Windows SDK, 3-75
Source and Interactive Execution windows, 5-35
User Interface Editor window, 4-23
Variables and Watch windows, 10-14
Help Style command, Options menu, 7-11 to 7-12
Hide command, File menu
Interactive Execution and Standard Input/Output windows (note), 5-7
Variables and Watch windows, 10-3
Hide Panels command, View menu, 4-12
Hide Windows option, Run Options command, 3-69
hierarchy buttons, Edit Menu Bar dialog box, 4-7
HKEY_LOCAL_MACHINE\SOFTWARE\National Instruments\CVI Run-Time Engine\cvirte, 1-2
HKEY_LOCAL_MACHINE\SOFTWARE\National Instruments\CVI\[version], 1-2
Horizontal Centers option
Alignment command, 4-13
Distribution command, 4-14
Horizontal Compress option, Distribution command, 4-14
Horizontal Gap option, Distribution command, 4-14

I

Icon control, Target Settings dialog box, 3-12
icons
associated with variables, 10-2
Project window, 3-1 to 3-2
Image base address compiler option, 3-63
Import Library Base Name option, Target Settings dialog box, 3-15
Import Library Choices button, Target Settings dialog box, 3-18
include directory (table), 1-3
Include File command, View menu
Function Panel Editor, 8-13
Function Panel windows, 6-15
Include File in Build command, Edit menu, 3-7
include files
adding missing files, 5-21
generating for Visual Basic, 5-34
prompting for path, 3-63
tracking dependencies, 3-63
Include option, Add Files to Project command, 3-7

Include Paths command, Options menu, 3-68
Initial Control Width command, Options method, 8-19
Input command
Create menu, 8-7
File menu, 11-4
input control parameters, specifying, 6-5
Insert Construct command, Edit menu, 5-9 to 5-10
Insert Function Call command, Code menu, 6-14
Insert Include Statements command, Build menu, 5-21
Insert New Item option, Edit Menu Bar dialog box, 4-7
Insert Separator option, Edit Menu Bar dialog box, 4-7
instrsup.dll
target settings for DLLs, 3-16 to 3-17
cvi_lvrt.dll subset, 3-17
libraries supported by, 3-16
Utility Library functions contained in, 3-16 to 3-17
target settings for executables, 3-12 to 3-13
libraries supported by, 3-12 to 13
Utility Library functions contained in, 3-13
Instrument command
adding instrument drivers to project, 3-7
Create command, Function Tree Editor, 7-5
Instrument Directories command, Options menu, 3-68
instrument driver function panel (.fp) files
adding to project list, 6-7
creating help information, 9-5 to 9-6
dummy .fp files for support libraries, 3-72
File menu commands
Function Panel Editor, 8-2 to 8-3
Function Tree Editor, 7-2 to 7-4
purpose and use, 3-1
Instrument Driver Only option
Run-Time Engine Support, Create Distribution Kit dialog box, 3-24 to 3-25
Runtime Support option, Target Settings dialog box
Target Type Dynamic Link Library, 3-15
Target Type Executable, 3-12
instrument drivers
compared with user libraries, 3-72
data types, 3-41 to 3-47
overview, 3-41 to 3-42
predefined, 3-42 to 3-44
user-defined, 3-44 to 3-45
VISA data types, 3-46
definition, 3-36
documenting, 3-50
files for instrument drivers, 3-36 to 3-37
forcing source code for loaded driver into Interactive window, 3-70
functions, 3-40 to 3-41
building function panels, 3-48 to 3-49
building function tree, 3-48
defining function parameters, 3-41
defining hierarchy of functions, 3-41
required functions, 3-47 to 3-48
structuring functions, 3-40 to 3-41
writing function code, 3-49
help information, 9-3
input and output parameters, 3-47
IVI instrument drivers
creating, 3-53, 5-29
editing attributes, 5-30
loading/unloading, 3-37 to 3-38
instruments without instrument program, 3-38
Load command, Instrument menu, 3-50, 7-7, 8-15

precedence rules, 3-37 to 3-38
Unload command, Instrument menu, 3-50, 7-8, 8-15
modifying, 3-39
modules containing non-instrument functions, 3-38
operating, 3-49
programming, 3-1
return values, 3-47
testing, 3-49
VXIplug&play instrument driver files, 3-37
Instrument menu
accessing function panels, 6-2
Function Panel Editor, 8-15 to 8-16
Edit command, 8-16
Load command, 8-15
Unload command, 8-15
Function Panel windows, 6-17
Function Tree Editor, 7-7 to 7-8
Edit command, 7-8
Load command, 7-7
Unload command, 7-8
Project window, 3-50 to 3-52
accessing function panels, 3-51 to 3-52
Edit command, 3-51
Load command, 3-50
Unload command, 3-50
Source and Interactive Execution windows, 5-28
Intelligent Virtual Instrument drivers. *See* IVI instrument drivers.
Interactive Execution command, Window menu, 3-61
Interactive Execution window, 5-3 to 5-4
Build menu, 5-18 to 5-21
Edit menu, 5-7 to 5-14
excluding lines, 5-9
executing code, 5-3 to 5-4
rules for, 5-4
File menu, 5-6 to 5-7
force loaded instrument driver's source into, 3-70
force project compiled source files into, 3-71
Instrument menu, 5-28
Interactive Window Memory Size control, 3-70
Library menu, 5-28
Options menu, 5-31 to 5-35
purpose and use, 2-3
rules for executing code, 5-4
Run menu, 5-22 to 5-28
selecting text, 5-4 to 5-6
subwindows, 5-4
Tools menu, 5-29 to 5-30
View menu, 5-14 to 5-18
Window menu, 5-31
Interpret As command, Options menu, 10-13
intrinsic C data types, 3-42 to 3-43
Item option, Edit Menu Bar dialog box, 4-6
IVI instrument drivers
creating, 3-53, 5-29
editing attributes, 5-30

K

keyboard commands for Source window (table), A-1 to A-2

L

Label Appearance section, Edit Control dialog box, 4-9
Labeling tool, 4-1
Label/Value Pairs button, 4-9
LabVIEW Real-Time Only option, Create Distribution Kit dialog box, 3-25 to 3-26

LabWindows/CVI. *See also* specific windows.
components, 2-1 to 2-4
LabWindows/CVI environment, 2-3 to 2-4
list of components, 2-1
standard libraries, 2-2
configuration options, 1-2 to 1-5
creating applications, 2-4 to 2-5
environment, 2-3 to 2-4
startup options (table), 1-1
Last Function Panel Window command, View menu
Function Panel Editor, 8-15
Function Panel windows, 6-16
Last Panel command, View menu, 6-2
Left Edges option
Alignment command, 4-13
Distribution command, 4-14
libraries
files required in project file list, 3-1
standard libraries, 2-2
static libraries
creating, 3-10, 5-20
target settings, 3-20
user libraries, 3-72
Library File option, Target Settings dialog box, 3-20
Library Generation Choices option, Target Settings dialog box, 3-20
Library menu
Function Panel windows, 6-17
Project window, 3-52
Source and Interactive Execution windows, 5-28
User Interface Editor, 4-21
Library option, Add Files to Project command, 3-6
Library Options command, Options menu, 3-71
Library Options dialog box
dummy .fp files for support libraries, 3-72
National Instrument Libraries, 3-71 to 3-72
User Libraries section, 3-72
Line command, View menu, 5-15
Line Icons command, View menu, 5-15, 5-23
Line Numbers command, View menu, 5-14
Line Select mode, 5-5
Line Terminator option, Editor Preferences command, 5-31
lines of code, excluding, 5-9
LINK_CVI_LVRT_ macro, 3-67
LINK_CVIRTE_ macro, 3-67
LINK_INSTRSUP_ macro, 3-67
Load command, Instrument menu
Function Panel Editor, 8-15
Function Tree Editor, 7-7
Project window, 3-50
Load From Text Format command, Options menu, 4-23
LoadExternalModule option
External Compiler support dialog box, 3-21
Target Settings dialog box
Target Type Dynamic Link Library, 3-18 to 3-19
Target Type Executable, 3-14
loading/unloading instrument drivers, 3-37 to 3-38
instruments without instrument program, 3-38
Load command, Instrument menu, 3-50, 7-7, 8-15
precedence rules, 3-37 to 3-38
Unload command, Instrument menu, 3-50, 7-8, 8-15
Low-Level Support Driver control, Create Distribution Kit dialog box, 3-27

M

_M_IX86 macro, 3-67
macros
  browsing, 3-56
  predefined, 3-67
Main Function command, Generate menu, 4-17 to 4-18
manual. *See* documentation.
Mark All for Compilation command, Build menu, 3-21
Mark File for Compilation command, Build menu
  Project window, 3-20
  Source and Interactive Execution windows, 5-20
Maximum number of compile errors option, 3-66
maximum stack size, setting, 3-62 to 3-63
Memory Display command
  View menu
    Array window, 11-6
    Variables and Watch windows, 10-8
  Window menu, 3-61
menu bar, Function Panel Editor, 8-1 to 8-2
Menu Bar command, Create menu, 4-10
Menu Bar Constant Prefix option, Edit Menu Bar dialog box, 4-6
Menu Bars command, Edit menu, 4-6 to 4-7
Menu Bars List dialog box, 4-6
Menu Callbacks command, Generate menu, 4-19
MenuFontBold option, 1-5
MenuFontName option, 1-5
MenuFontSize option, 1-5
Message command, Create menu, 8-12
meta data types
  Any Array, 3-43
  Any type, 3-43 to 3-44
  Numeric Array, 3-43
  Var Args, 3-44
Microsoft Visual Basic, generating include file for, 5-34
Minimize All command, Window menu, 3-60
Modifier Key, Edit Menu Bar dialog box, 4-6
Move Backward option, Control ZPlane Order command, 4-15
Move Cursor to the End of Pasted Text option, Editor Preferences command, 5-31
Move Forward option, Control ZPlane Order command, 4-15
Move Item Down command, Edit menu, 3-7
Move Item Up command, Edit menu, 3-7
Move to Back option, Control ZPlane Order command, 4-15
Move to Front option, Control ZPlane Order command, 4-15
multi-dimensional arrays
  Array Display window, 11-2 to 11-3
  illustration, 11-2
  Reset Indices dialog box, 11-3 to 11-4
  specifying dimensions, 11-3 to 11-4
  String Display window, 11-3 to 11-4
multi-dimensional string array, 11-3 to 11-4
Multiple Files option, Find command, 5-14

N

Name option, Find command, 5-13, 10-4
New command, File menu
  Array and String Display windows, 11-4
  creating help information, 9-5
  Function Panel Editor, 8-2
  Function Panel windows, 6-7
  Function Tree Editor, 7-2
  invoking Function Tree Editor, 7-1
  Project window, 3-3
  Source and Interactive Execution windows, 5-6
  User Interface Editor, 4-3
  Variables and Watch windows, 10-3

New Window option, Select Function Panel dialog box, 3-51
-newproject option (table), 1-1
Next Build Error command, View menu, 5-21
Next File command, Edit menu, 5-14
Next Function Panel command, View menu, 6-16
  Function Panel Editor, 8-14
Next Function Panel Window command, View menu
  Function Panel Editor, 8-14 to 8-15
  Function Panel windows, 6-16
Next Panel command, View menu
  Function Panel windows, 6-2
  User Interface Editor window, 4-12
Next Scope command, Edit menu, 10-5
Next Tag command, View menu, 5-15
Next Tool command, Options menu, 4-21
NI Developer Zone, B-1
NI Reports Support control, Create Distribution Kit dialog box, 3-26
_NI_BC macro, 3-67
_NI_i386_ macro, 3-67
_NI_mswin_ macro, 3-67
_NI_mswin32_ macro, 3-67
_NI_SC macro, 3-67
_NI_VC macro, 3-67
No Sorting command, View menu, 3-8
non-void functions, requiring return values for, 3-65 to 3-66
__NT__ macro, 3-63
NUL byte, difference from space character (note), 10-1, 11-1
Numeric Array data type, 3-43
Numeric command, Create menu, 8-10 to 8-11
numeric control parameters, specifying, 6-5

O

'O' option, making compatible with CVI 5.0.1, 3-64
object files
  creating, 5-34
  required in project file list, 3-1
Object option, Add Files to Project command, 3-6
ODL file, generating, 7-12
one-dimensional array
  displaying in Array Display window (figure), 11-2
  Graphical Array view (figures), 10-9 to 10-10
Open command, File menu
  Array and String Display windows, 11-4
  creating help information, 9-5
  Function Panel Editor, 8-2
  Function Panel windows, 6-7
  Function Tree Editor, 7-2
  invoking Function Tree Editor, 7-1
  Project window, 3-4
  Source and Interactive Execution windows, 5-6
  User Interface Editor, 4-3
  Variables and Watch windows, 10-3
Open Quoted Text command, File menu, 5-6
Operate Function Panel command, Options menu, 8-19
Operate Visible Panels command, Options menu, 4-21
Operating tool, 4-1
Options menu
  Array and String Display windows
    Display Entire Buffer command, 11-8
    Reset Indices command, 11-2 to 11-3, 11-8

Function Panel Editor, 8-18 to 8-19
Data Types command, 8-18
Edit Function Tree command, 8-19
Initial Control Width command, 8-19
Operate Function Panel command, 8-19
Panels Movable command, 8-19
Revert to Default Panel Size command, 8-19
Toggle Scroll Bars command, 8-19
Toolbar command, 8-19
Function Panel windows, 6-18 to 6-19
Change Format command, 6-19
Default All command, 6-18
Default Control command, 6-18
Edit Function Panel Window command, 6-19
Exclude Function command, 6-18
Toggle Control Style command, 6-5, 6-18
Toolbar command, 6-18
Function Tree Editor, 7-11 to 7-14
Create DLL Project command, 7-13
.FP File Format command, 7-11
Generate DEF File command, 7-12
Generate Documentation command, 7-12
Generate Function Prototypes command, 7-12
Generate ODL File command, 7-12
Generate Windows Help command, 7-12
Help Style command, 7-11 to 7-12
Transfer Window Help to Function Help command, 7-12
VXIplug&play Style command, 7-13 to 7-14
Project window
Build Options command, 3-62 to 3-66
Colors command, 3-75
Command Line command, 3-69
Compiler Defines command, 3-66 to 3-67
Environment command, 3-51, 3-69 to 3-71
Font command, 3-74
Include Paths command, 3-68
Instrument Directories command, 3-68
Library Options command, 3-71 to 3-72
Run Options command, 3-68 to 3-69
Source Code Control Options, 3-73 to 3-74
Tools Menu Options command, 3-72 to 3-73
Source and Interactive Execution windows
Bracket Styles command, 5-32
Colors command, 5-32
Create Object File command, 5-34
Editor Preferences command, 5-36
Font command, 5-32
Generate DLL Import Library command, 5-34
Generate DLL Import Source command, 5-33
Generate Visual Basic Include command, 5-34
Help menu, 5-35
Preprocess Source File command, 5-35
Syntax Coloring option, 5-32
Toolbar command, 5-31
Translate DOS LW program command, 5-33
User Defined Tokens for Coloring command, 5-33
User Interface Editor
Assign Missing Constants command, 4-23

Load From Text Format command, 4-23
Next Tool command, 4-21
Operate Visible Panels command, 4-21
Preferences command, 4-22 to 4-23
Save in Text Format command, 4-23
Variables and Watch windows
Add Watch Expression command, 10-2, 10-14
Estimate Number of Elements command, 10-14
Interpret As command, 10-13
Variable Size command, 10-13
Other User Interface Editor Preferences dialog box, 4-23
Output command
Create menu, 8-11
File menu
Array and String Display windows, 11-4
Variables and Watch windows, 10-3
output control parameters, specifying, 6-6
Overwrite command, Edit menu, 11-6

P

Panel Callback command, Generate menu, 4-18
Panel command
Create menu, 4-10
Edit menu, 4-7 to 4-8
Panel Settings section, Edit Panel dialog box, 4-7 to 4-8
panels
copying or cutting, 4-5
Edit Panel dialog box, 4-7 to 4-8
preferences for new panels, 4-22
showing/hiding, 4-12
Panels Movable command, Options menu, 8-19
parent structure, 10-2
parentheses, finding pairs of, 5-10
Paste command, Edit menu
adding help information, 9-6
Function Panel Editor, 8-4
Source and Interactive Execution windows, 5-8
User Interface Editor window, 4-5
Paste Above command, Edit menu, 7-3
Paste Below command, Edit menu, 7-3
path options, Prompt for include file paths, 3-63
pathnames
Show Full Pathnames command, View menu, 3-8
Sort by Pathnames command, View menu, 3-8
paths for compiler, listing, 3-68
pointer mismatch warning, detecting, 3-64
pop-up menus, User Interface Editor window, 4-2
-pProcessID option (table), 1-2
predefined data types, 3-42 to 3-44
intrinsic C data types, 3-42 to 3-43
meta data types, 3-43 to 3-44
Any Array, 3-43
Any type, 3-43 to 3-44
Numeric Array, 3-43
Var Args, 3-44
predefined macros, 3-67
Preferences command
Code menu, 4-20
Always Append Code to End option, 4-20
Default Control Events option, 4-20
Default Panel Events option, 4-20
Options menu, 4-22 to 4-23
More button, 4-23
Preferences for New Controls section, 4-23

Preferences for New Panels section, 4-22
User Interface Editor Preferences section, 4-22
Preprocess Source File command, Options menu, 5-35
Preview User Interface Header File command, View menu, 4-12
Previous Build Error command, View menu, 5-21
Previous Function Panel command, View menu
Function Panel Editor, 8-14
Function Panel windows, 6-16
Previous Function Panel Window command, View menu
Function Panel Editor, 8-14
Function Panel windows, 6-16
Previous Panel command, View menu
Function Panel windows, 6-2
User Interface Editor window, 4-12
Previous Scope command, Edit menu, 10-5
Previous Tag command, View menu, 5-15
Print command, File menu
Project window, 3-5
Source and Interactive Execution windows, 5-7
User Interface Editor window, 4-4
Project command, Window menu, 3-60
project files
optional files, 3-1
required files, 3-1
saving automatically, 3-5
View menu commands for displaying, 3-8
Project window
Build menu, 3-8 to 3-30
debugging DLLs, 3-31 to 3-35
Edit menu, 3-6 to 3-7
File menu, 3-3 to 3-5
Help menu, 3-75 to 3-76
icons, 3-1 to 3-2
Instrument menu, 3-50 to 3-52
Library menu, 3-52
opening
with New command, 3-3
with Open command, 3-4
optional files, 3-1
Options menu, 3-62 to 3-75
overview, 3-1 to 3-3
purpose and use, 2-3
required files, 3-1
Run menu, 3-31 to 3-35
selecting multiple files, 3-2 to 3-3
Tools menu, 3-52 to 3-59
using instrument drivers, 3-36 to 3-50
View menu, 3-8
Window menu, 3-59 to 3-62
Prompt for include file paths option, 3-63
Properties command, Source Code Control submenu, 3-54
Purge Undo Actions When Saving File option, Editor Preferences command, 5-31

Q

Quick Edit Window
Edit Control dialog box, 4-9
Edit Panel dialog box, 4-8

R

Read Only command, File menu
creating help information, 9-6
Function Panel Editor, 8-3
Function Tree Editor, 7-3
Source and Interactive Execution windows, 5-7
User Interface Editor, 4-4
Reattach Program option, Edit Instrument dialog box, 3-51, 7-8

Recall Function Panel command, View menu
invoking, 5-16
multiple functions in one function panel window, 5-17
multiple panels for one function, 5-17
purpose, 5-16
recalling from function name only, 5-17
syntax requirements, 5-17
Redo command, Edit menu
adding help information, 9-6
Function Panel Editor, 8-3 to 8-4
Source and Interactive Execution windows, 5-8
User Interface Editor window, 4-4
Refresh Status command, Source Code Control submenu, 3-54
regular expression characters (table), 5-12 to 5-13
Regular Expression option, Find command
Source and Interactive Execution windows, 5-11 to 5-13
Variables window, 10-4
Regular Expression option, Find UIR Objects dialog box, 4-11
Release option, Configuration command, 3-9
Remove File command, Edit menu, 3-7
Remove From Source Control command, Source Code Control submenu, 3-54
Replace command, Edit menu
creating help information, 9-7
Find Next button, 5-14
Function Panel Editor, 8-6
Function Tree Editor, 7-5
Replace All button, 5-14
Return button, 5-14
Stop button, 5-14
Require function prototypes option, 3-65, 5-4
Require return values for non-void functions option, 3-65 to 3-66
Reset Indices command, Options menu
description, 11-8
displaying multi-dimensional arrays, 11-3 to 11-4
displaying single-dimensional arrays, 11-2
specifying index for string array, 11-3
specifying plane for multi-dimensional arrays, 11-2
Reset Indices dialog box, 11-3 to 11-4
Resolve All Excluded Lines command, Edit menu, 5-9
Retrace Pointer Chain command, View menu, 10-2, 10-7
Return Value command, Create menu, 8-11 to 8-12
return value controls parameter, specifying, 6-4
return values, requiring for non-void functions, 3-65 to 3-66
Revert command, Edit menu, 9-7
Revert to Default Panel Size command, Options menu, 8-19
Right Edges option
Alignment command, 4-13
Distribution command, 4-14
Ring command, Create menu, 8-9
Run Function Panel command, Code menu, 6-8
Run Interactive Statements command, Run menu, 5-24 to 5-25
Run menu
Array and String Display windows, 11-7
Project window
Break at First Statement command, 3-34
Breakpoints command, 3-34 to 3-35
Continue command, 3-34
Debug command, 3-31
Execute command, 3-35
Select External Process command, 3-33
Terminate Execution command, 3-34

Threads command, 3-35
Source and Interactive Execution windows
Add Watch Expression command, 5-27
Break at First Statement command, 5-23, 5-26
Breakpoints command, 5-23, 5-26 to 5-27
Continue command, 5-25
Debug/Run Interactive Statements command, 5-24 to 5-25
Down Call Stack command, 5-27
Finish Function command, 5-25
Go To Cursor command, 5-25
Run Interactive Statements command, 5-24 to 5-25
Stack Trace command, 5-27
Step Into command, 5-25
Step Over command, 5-25
Terminate Execution command, 5-26
Threads command, 5-28
Toggle Breakpoint command, 5-23, 5-26
Up Call Stack command, 5-27
View Variable Value command, 5-27, 10-1, 11-1
User Interface Editor, 4-20
Variables and Watch windows, 10-12 to 10-13
Run Options command, Options menu, 3-68 to 3-69
Break On First Chance Exceptions, 3-68 to 3-69
Break on library errors option, 3-68
Enable global Ctrl+F12 debug break key, 3-69
Hide windows, 3-69
Save changes before running, 3-68
-run startup option (table), 1-1
-run_then_exit startup option (table), 1-1
Run-Time Engine Support, Create Distribution Kit dialog box, 3-24 to 3-25
All Engines option, 3-26
Full Runtime Engine option, 3-24
Instrument Driver Only option, 3-24 to 3-25
LabVIEW Real-Time Only option, 3-25 to 3-26
None option, 3-26
run-time error reporting
Run menu, Project window, 3-31
Source and Interactive Execution windows, 5-25
Run-Time Errors command, Window menu, 3-60
Runtime Support option, Target Settings dialog box
Target Type Dynamic Link Library, 3-15 to 3-17
Full Runtime Engine option, 3-15
instrsup.dll, 3-16 to 3-17
Instrument Driver Only option, 3-15
Target Type Executable, 3-12 to 3-13
Full Runtime Engine option, 3-12
instrsup.dll, 3-12 to 3-13
Instrument Driver Only option, 3-12

S

Save command, File menu
Array and String Display windows, 11-4
creating help information, 9-5
Function Panel Editor, 8-2
Function Panel windows, 6-7
Function Tree Editor, 7-2
Project window, 3-4
Source and Interactive Execution windows, 5-6
User Interface Editor, 4-3
Variables and Watch windows, 10-3

Save All command, File menu
Array and String Display windows, 11-4
creating help information, 9-5
Function Panel Editor, 8-3
Function Panel windows, 6-7
Function Tree Editor, 7-3
Project window, 3-5
Source and Interactive Execution windows, 5-7
User Interface Editor, 4-3
Variables and Watch windows, 10-3
Save As command, File menu
Source and Interactive Execution windows, 5-6
User Interface Editor, 4-3
Save changes before running option, Run Options command, 3-68
Save Copy As command, File menu
Source and Interactive Execution windows, 5-7
User Interface Editor window, 4-3
Save Copy of .FP File command, File menu
adding help information, 9-5
Function Panel Editor, 8-2
Function Tree Editor, 7-2
Save .FP File command, File menu
adding help information, 9-5
Function Panel Editor, 8-2
Function Tree Editor, 7-2
Save .FP File As command, File menu
adding help information, 9-5
Function Panel Editor, 8-2
Function Tree Editor, 7-3
Save in Text Format command, Options menu, 4-23
Save Project As command, File menu, Project window, 3-4
sdk directory (table), 1-3
Search By option, Find UIR Objects dialog box, 4-11
Select All command, Edit menu
Project window, 3-7
Source and Interactive Execution windows, 5-9
Select Attribute Constant command, Code menu
attribute constants, 6-10 to 6-11
attribute values, 6-11
Select Attribute Constant dialog box, 6-10 to 6-11
Select Attribute Values dialog box, 6-11
Select External Process command, Run menu, 3-33
Select Function Panel dialog box, 3-51 to 3-52
Alphabetize command, 3-51
Flatten checkbox, 3-51, 6-2
Function Names command, 3-51
Help button, 3-52
New Window command, 3-51
Select UIR Constant command, Code menu, 6-9 to 6-10
Select UIR Constant dialog box, 6-9 to 6-10
Select Variable or Expression command, Code menu, 6-12 to 6-14
Select Variable or Expression dialog box, 6-12 to 6-14
Data Type, 6-12
data type compatibility, 6-13
Data Type of Control, 6-12
items included in list box, 6-12
Show Project Variables option, 6-12
sorting of list box entries, 6-14
Variable or Expression list box, 6-12
separators, adding and positioning on toolbar, 5-2
Set Active Project command, File menu, Project window, 3-4
Set Default Font command, Edit menu, 4-10
Set Target File command, Code menu
Function Panel windows, 6-14
User Interface Editor window, 4-15
Set Target File dialog box, 4-15

Shortcut Key, Edit Menu Bar dialog box, 4-6
shortcut options, Create Distribution Kit dialog box, 3-28 to 3-29
Show Build Error window for warnings option, 3-66
Show Differences command, Source Code Control submenu, 3-54
Show Full Dates command, View menu, 3-8
Show Full Path Names command, View menu, 3-8
Show History command, Source Code Control submenu, 3-54
Show Info command, Edit Instrument dialog box
  Function Tree Editor, 7-8
  Project window, 3-51
Show/Hide Panels command, View menu, 4-12
signed/unsigned pointer mismatches, detecting, 3-64
single-dimensional array, displaying in Array Display window (figure), 11-2
skeleton code
  definition, 2-5, 4-2
  function skeletons, 4-17
  placement in target file, 4-17
Sleep Policy, CVI Environment option, 3-68 to 3-69
Slide command, Create menu, 8-8
slide controls, specifying parameters, 6-5
Sort By Date command, View menu, 3-8
Sort By File Extension command, View menu, 3-8
Sort By Name command, View menu, 3-8
Sort By Pathname command, View menu, 3-8
Source Code Browser, Tools menu, 3-55 to 3-56
  files, 3-55
  functions, 3-56
  generating browse information, 3-66
  variables, data types, and macros, 3-56
Source Code Browser command, View menu
  Array Display window, 11-6
  Variables and Watch windows, 10-8
Source Code Connection
  Edit Control dialog box, 4-8
  Edit Panel dialog box, 4-7
Source Code Control command, Tools menu
  Function Tree Editor, 7-10
  Source and Interactive Execution windows, 5-30
  Source Code Control submenu, 3-55
Source Code Control Errors window, 3-61
Source Code Control Options command, 3-73
Source Code Control Options dialog box, 3-73 to 3-74
  Advanced, 3-74
  Always show confirmation dialog, 3-74
  Attach, 3-73
  Create, 3-73
  Do not include .prj file in SCC actions, 3-74
  Perform same actions for .h file as for .uir file, 3-74
  Project, 3-73
  Provider, 3-73
  Suppress CVI error messages, 3-72
  Use default comment, 3-74
  Use default username, 3-74
  Use global source code control settings, 3-73
  Use project source code control settings, 3-73
Source Code Control submenu, Tools menu, 3-53 to 3-55
  Add File to Source Control, 3-54
  Check In, 3-54
  Check Out, 3-54
  Clear Source Code Control Error Window, 3-55
  Get Latest Version, 3-54
  Get Latest Version of All, 3-54

Properties, 3-54
Refresh Status, 3-54
Remove From Source Control, 3-54
Show Differences, 3-54
Source Code Control, 3-55
Undo Check Out, 3-54
source files
creating with CodeBuilder, 4-2 to 4-3
debugging, 2-4
forcing into Interactive window
loaded instrument driver, 3-70
project compiled source files, 3-71
listed in Window menu, 3-62
preprocessing, 5-35
required in project file list, 3-1
Source option, Add Files to Project command, 3-6
Source window
Build menu, 5-18 to 5-21
context menus, 5-3
Edit menu, 5-7 to 5-14
File menu, 5-6 to 5-7
Instrument menu, 5-28
keyboard commands (table), A-1 to A-2
Library menu, 5-28
notification of external modification, 5-2
opening
with New command, 3-3
with Open command, 3-4
Options menu, 5-31 to 5-35
purpose and use, 2-3, 5-1
Run menu, 5-22 to 5-28
selecting text, 5-4 to 5-6
subwindows, 5-4
Tools menu, 5-29 to 5-30
View menu, 5-14 to 5-18
Window menu, 5-31
space character, difference from NUL byte (note), 10-1, 11-1
stack size, setting, 3-62 to 3-63
Stack Trace command, Run menu, 5-27
standalone executables, creating and distributing. *See also* Create Distribution Kit dialog box.
Create Debuggable Executable command, 3-9, 5-19
Create Release Executable command, 3-10, 5-19 to 5-20
Runtime Support option, Target Settings dialog box
Target Type Dynamic Link Library, 3-15 to 3-17
Target Type Executable, 3-12 to 3-13
standard libraries, 2-2
startup options for LabWindows/CVI (table), 1-1
static libraries
creating, 3-10, 5-20
target settings, 3-20
status dialog box, displaying, 3-64
Step Into command, Run menu, 5-25
Step Over command, Run menu, 5-25
Stop on first file with errors option, 3-66
String Display command, View menu, 11-3, 11-6
String Display window
Edit menu, 11-5 to 11-6
File menu, 11-4
Format menu, 11-7
Help menu, 11-8
multi-dimensional strings, 11-3 to 11-4
Options menu, 11-8
purpose and use, 2-4, 11-3
Run menu, 11-7
Window menu, 11-7
structures
child structure, 10-2
Follow Pointer Chain command, 10-7
parent structure, 10-2
pointer-linked structures, 10-7

replacing, 10-7
Retrace Pointer Chain command, 10-7
subwindows, in Source and Interactive Execution windows, 5-4
symbols
options for exporting symbols in DLLs, 3-19
specifying in External Compiler Support dialog box, 3-22
Syntax Coloring option, Options menu, 5-32
system colors, selecting for panels, 4-22
system integration, by National Instruments, B-1

T

Tab Length option, Editor Preferences command, 5-31
Tab Order command, Edit menu, 4-9
Tag Scope command, View menu, 5-16
tagged lines
Clear Tags command, 5-16
Next Tag command, 5-15
Previous Tag command, 5-15
Tag Scope command, 5-16
Toggle Tag command, 5-15
Target Settings command, Build menu, 3-12
Target Settings dialog box, 3-12 to 3-20
Target Type Dynamic Link Library, 3-15 to 8-20
DLL file, 3-15
Exports, 3-19
Import Library Base Name, 3-15
Import Library Choices button, 3-18
Runtime Support, 3-15 to 3-17
Type Library, 3-18
Using LoadExternalModule, 3-18 to 3-19
Version Info, 3-18
Where to Copy DLL, 3-15
Target Type Executable, 3-12 to 3-15
Application file, 3-12
Application Icon File, 3-12
Application Title, 3-12
Create Console Application, 3-14
Icon, 3-12
LoadExternalModule Options, 3-14
Runtime Support, 3-12 to 3-13
Version Info, 3-14
Target Type Static Library
Library File, 3-20
Library Generation Choices, 3-20
Target Type command, Build menu, 3-11
technical support resources, B-1 to B-2
Terminate Execution command, Run menu
Project window, 3-34
Source and Interactive Execution windows, 5-26
text, selecting
Character Select mode, 5-5
Column Select mode, 5-6
Line Select mode, 5-5
text format
Load From Text Format command, 4-23
Save In Text Format command, 4-23
Threads command, Run menu
Project window, 3-35
Source and Interactive Execution windows, 5-28
Tile Windows command, Window menu, 3-59
time option, DSTRules, 1-4
timer option, useDefaultTimer, 1-4
Tip of the Day command, Help menu, 3-76
tmpdir configuration option, 1-3
Toggle Breakpoint command, Run menu, 5-23, 5-26
Toggle Control Style command, Options menu
description, 6-18
specifying binary control parameter, 6-6
specifying numeric control parameter, 6-5
specifying slide control parameter, 6-5

Toggle Exclusion command, Edit menu, 5-3, 5-9
Toggle Scroll Bars command, Options menu, 8-19
Toggle Tag command, View menu, 5-15
tokens
  Syntax Coloring option, Options menu, 5-32
  User Defined Tokens for Coloring command, Options menu, 5-33
Toolbar command
  Options menu
    Function Panel Editor, 8-19
    Function Panel windows, 6-18
    Source and Interactive Execution windows, 5-31
  View menu
    Function Panel Editor, 8-13
    Function Panel windows, 6-15
    Source and Interactive Execution windows, 5-15
toolbars, 5-1 to 5-2
  adding and positioning buttons and separators, 5-2
  displaying names of button or icons, 5-1
  function panels, 6-4
  modifying, 5-1 to 5-2
  positioning controls, 5-2
  removing items, 5-2
Tools menu
  creating help information, 9-7
  Function Panel Editor, 8-16 to 8-18
    Create ActiveX Controller command, 8-16
    Create ActiveX Server command, 8-16
    Create IVI Instrument Driver command, 8-17
    Edit ActiveX Server command, 8-17
    Edit Instrument Attributes command, 8-17
    Enable Auto Replace command, 8-17
    Generate Source For Function Panel command, 8-17
    Go To Declaration command, 8-18
    Go To Definition command, 8-17
Function Panel windows, 6-17
Function Tree Editor, 7-9 to 7-10
  Create ActiveX Controller command, 7-9
  Create ActiveX Server command, 7-9
  Create IVI Instrument Driver command, 7-9
  Edit ActiveX Server command, 7-9
  Edit Instrument Attributes command, 7-9
  Enable Auto Replace command, 7-10
  Generate IVI C++ Wrapper command, 7-9
  Generate New Source For Function Tree command, 7-10
  Go To Declaration command, 7-10
  Go To Definition command, 7-10
  Source Code Control command, 7-10
Project window, 3-52 to 3-59
  Convert UI to Lab Style command, 3-59
  Create ActiveX Automation Controller command, 3-52
  Create ActiveX Server command, 3-53
  Create IVI Instrument Driver command, 3-53
  Edit ActiveX Server command, 3-53
  Source Code Browser command, 3-55 to 3-56
  Source Code Control, 3-53 to 3-55
  UI to Code Converter utility, 3-56 to 3-58
  User Interface Localizer utility, 3-58 to 3-59
  user-defined entries, 3-59

Source and Interactive Execution windows, 5-29 to 5-30
Create ActiveX Controller command, 5-29
Create IVI Instrument Driver command, 5-29
Edit ActiveX Server command, 5-29
Edit Function Panel command, 5-30
Edit Function Tree command, 5-30
Edit Instrument Attributes command, 5-30
Source Code Control command, 5-30
User Interface Editor, 4-21
Tools Menu Options command, 3-72 to 3-73
Tools Menu Options dialog box, 3-73
Top Edges option
Alignment command, 4-13
Distribution command, 4-14
Track include file dependencies option, 3-63
Transfer Window Help to Function Help command, Options menu, 7-12
Translate LW DOS program command, Options menu, 5-33
two-dimensional array, Graphical Array view (figure), 10-11 to 10-12
Type Library button, Target Settings dialog box, 3-18
Type option, Find command, 5-13, 10-4

U

UI to Code Converter utility, Tools menu, 3-56 to 3-58
generated code, 3-57 to 3-58
limitations, 3-58
using, 3-56 to 3-57
UIR Callbacks Object File option, External Compiler Support dialog box, 3-21
.uir files. *See* user interface resource (.uir) files.
Undo command, Edit menu
adding help information, 9-6
Function Panel Editor, 8-3 to 8-4
Source and Interactive Execution windows, 5-8
User Interface Editor window, 4-4
Undo Checkout command, Source Code Control submenu, 3-54
Undoable Actions Per File (Next Session) option, Editor Preferences command, 5-31
unitialized variable options
Detect unitialized local variables at run-time, 3-63
Unitialized local variables detection
Aggressive mode, 3-64
Conservative mode, 3-64
Disabled mode, 3-64
Unload command, Instrument menu
Function Panel Editor, 8-15
Function Tree Editor, 7-8
Project window, 3-50
unloading instrument drivers. *See* loading/unloading instrument drivers.
unreachable code, detecting, 3-64 to 3-65
unreferenced identifiers, detecting, 3-65
Up Call Stack command, Run menu, 5-27
Use Only One Browse Info Window option, Environment command, 3-70
Use Only One Function Panel Window option, Environment command, 3-51, 3-70
useDefaultTimer option, 1-4
user-defined entries, Tools menu, 3-59
User Defined Tokens for Coloring command, Options menu, 5-33
user interface constants, selecting, 6-9 to 6-10
attribute constants, 6-10 to 6-11
attribute value, 6-11
from .uir files, 6-9 to 6-10
User Interface Editor
Arrange menu, 4-13 to 4-15
Code menu, 4-15 to 4-20
CodeBuilder overview, 4-2 to 4-3

Create menu, 4-10
Edit menu, 4-4 to 4-10
File menu, 4-3 to 4-4
Library menu, 4-21
Options menu, 4-21 to 4-23
overview, 4-1 to 4-2
Run menu, 4-20
tool icons, 4-1 to 4-2
Tools menu, 4-21
using pop-up menus, 4-2
View menu, 4-11 to 4-12
Window menu, 4-21
User Interface Editor Preferences dialog box
More button, 4-23
Preferences for New Controls section, 4-23
Preferences for New Panels section, 4-22
User Interface Editor Preferences section, 4-22
User Interface Editor window
Coloring tool, 4-1
Editing tool, 4-1
eyedropper tool, 4-2
Help menu, 4-23
Labeling tool, 4-1
moving to, using Find UI Object command, 5-18
opening
with New command, 3-3
with Open command, 3-4
Operating tool, 4-1
popup menus, 4-2
purpose and use, 2-3
tool bar, 4-1
User Interface Localizer utility, Tools menu, 3-58 to 3-59
user interface objects, finding, 4-11 to 4-12, 5-18
User Interface option, Add Files to Project command, 3-7
user interface resource (.uir) files
Convert UI to Lab Style command, 3-59
optional for project file list, 3-1
UI to Code Converter utility, 3-56 to 3-58
generated code, 3-57 to 3-58
limitations, 3-58
using, 3-56 to 3-57
UIR Callbacks option, External Compiler Support dialog box, 3-21
User Interface Localizer utility, 3-58 to 3-59
user libraries. *See also* libraries.
dummy .fp files for support libraries, 3-72
installing into Library menu, 3-72
instrument drivers vs., 3-72
specifying in Library Options dialog box, 3-72
user-defined data types, 3-44 to 3-45
array data types, 3-45
creating, 3-45
Using LoadExternalModule to Load Object and Static Library Files option, External Compiler support dialog box, 3-21

V

Value option, Find command, 5-13, 10-4
Var Args data type, 3-44
Variable Size command, Options menu, 10-13
variables
browsing, 3-56
compiler options
Detect unitialized local variables at run-time, 3-63
Unitialized local variables detection, 3-64
Declare Variable dialog box, 6-8 to 6-9
Select Variable or Expression dialog box, 6-12 to 6-14
Variables command, Window menu
Project window, 3-61
Variables window, 10-1

Variables window, 10-1 to 10-14
Edit menu, 10-3 to 10-5
File menu, 10-3
Format menu, 10-12
Function subwindow, 10-1
Global subwindow, 10-1
Help menu, 10-14
icons associated with variables, 10-2
Options menu, 10-13 to 10-14
purpose and use, 2-4
Run menu, 10-12 to 10-13
View menu, 10-6 to 10-12
viewing, 10-1
Window menu, 10-13
Version Info button, Target Settings dialog box
Target Type Dynamic Link Library, 3-18
Target Type Executable, 3-14
Vertical Centers option
Alignment command, 4-13
Distribution command, 4-14
Vertical Compress option, Distribution command, 4-14
Vertical Gap option, Distribution command, 4-14
View command, Code menu, 4-19
View menu
Array Display window, 11-6 to 11-7
Graphical Array View command, 11-7
Memory Display command, 11-6
Source Code Browser command, 11-6
String Display command, 11-6
Function Panel Editor, 8-13 to 8-15
Current Tree command, 8-13
Error command, 8-13
Find Function Panel command, 8-14
First Function Panel Window command, 8-15
Function Panel History command, 8-14
Include File command, 8-13
Last Function Panel Window command, 8-15
Next Function Panel command, 8-14
Next Function Panel Window command, 8-14 to 8-15
Previous Function Panel command, 8-14
Previous Function Panel Window command, 8-14
Toolbar command, 8-13
Function Panel windows, 6-15 to 6-16
Current Tree command, 6-15
Error command, 6-15
Find Function Panel command, 6-15 to 6-16
First Function Panel Window command, 6-16
First Panel command, 6-2
Function Panel History command, 6-15
Include File command, 6-15
Last Function Panel Window command, 6-16
Last Panel command, 6-2
Next Function Panel command, 6-16
Next Function Panel Window command, 6-16
Next Panel command, 6-2
Previous Function Panel command, 6-16
Previous Function Panel Window command, 6-16
Previous Panel command, 6-2
Toolbar command, 6-15
Project window, 3-8
No Sorting command, 3-8
Show Full Dates command, 3-8

Show Full Path Names command, 3-8
Sort By Date command, 3-8
Sort By File Extension command, 3-8
Sort By Name command, 3-8
Sort By Pathname command, 3-8
Source and Interactive Execution windows
Beginning/End of Selection command, 5-15
Build Errors in Next File command, 5-21
Clear Tags command, 5-16
Find Function Panel command, 5-17 to 5-18
Find UI Object command, 5-19
Function Panel Tree command, 5-16
Line command, 5-15
Line Icons command, 5-15, 5-23
Line Numbers command, 5-14
Next Tag command, 5-15
Previous Tag command, 5-15
Recall Function Panel command, 5-16 to 5-17
Tag Scope command, 5-16
Toggle Tag command, 5-15
Toolbar command, 5-15
User Interface Editor
Find UIR Objects command, 4-11 to 4-12
Preview User Interface Header File command, 4-12
Show/Hide Panels command, 4-12
Variables and Watch windows
Array Display command, 10-8, 11-1
Close Variable command, 10-2, 10-7
Expand Variable command, 10-2, 10-7
Follow Pointer Chain command, 10-2, 10-7
Go To Definition command, 10-8
Go To Execution Position command, 10-8
Graphical Array View command, 10-9 to 10-12
Memory Display command, 10-8
Retrace Pointer Chain command, 10-2, 10-7
Source Code Browser command, 10-8
String Display command, 10-8, 11-3
View Variable Value command
Code menu
Array Display window, 11-1
Function Panel windows, 6-7, 6-14
String Display window, 11-3
Run menu
Array Display window, 11-1
Source and Interactive Execution windows, 5-27
String Display window, 11-3
Variables window, 10-1
VISA data types, 3-46
Visual Basic, generating include file for, 5-34
VXIplug&play instrument driver files, 3-37
VXIplug&play Style command, Options menu, 7-13 to 7-14

W

Watch command, Window menu
Project window, 3-61
Watch window, 10-2
watch variables/expressions
Add/Edit Watch Expression dialog box, 10-2
applicable only in source code modules (note), 5-22
purpose and use, 5-22 to 5-23
selecting, 10-2
suspending program execution conditionally, 5-23

Watch window, 10-1 to 10-14
activating, 10-2
Add/Edit Watch Expression dialog box, 10-2
Edit menu, 10-5 to 10-6
File menu, 10-3
Format menu, 10-12
Help menu, 10-14
purpose and use, 2-4, 10-2
selecting variables and expressions, 10-2
View menu, 10-6 to 10-12
Window menu, 10-13
Web Links command, Help menu, 3-76
Web support from National Instruments, B-1
Where to Copy DLL, Target Settings dialog box, 3-15
Whole Word option
Find command
Source and Interactive Execution windows, 5-11
Variables window, 10-4
Find UIR Objects dialog box, 4-11
WIN32 macro, 3-63
_WIN32 macro, 3-63
__WIN32__ macro, 3-63
Win95 DCOM/RTE Support option, Create Distribution Kit dialog box, 3-27
Window Help command, Edit menu, Function Panel Editor, 8-7
Window menu
Array Display window, 11-7
creating help information, 9-7
Function Panel Editor, 8-18
Function Panel windows, 6-17
Function Tree Editor, 7-10
Project window
Build Errors command, 3-60
Cascade Windows command, 3-59
Close All command, 3-60
Debug Output, 3-60
Interactive Execution command, 3-61
Memory Display command, 3-61
Minimize All command, 3-60
open source files, 3-62
Project command, 3-60
Run-Time Errors command, 3-60
Source Code Control Errors window, 3-61
Tile Windows command, 3-59
Variables command, 3-61
Watch command, 3-61
Source and Interactive Execution windows, 5-31
String Display window, 11-7
User Interface Editor, 4-21
Variables window, 10-1, 10-13
Watch window, 10-2, 10-13
windows, hiding, 3-69
Windows DLLs. *See* DLLs.
_WINDOWS macro, 3-63
Windows SDK command, Help menu, 3-75
WinMain, using instead of main, 4-18
workspace
creating, with New command, 3-3
Edit Workspace dialog box, 3-6
opening
with Open command, 3-4
with Set Active Project command, 3-4
Workspace command, Edit menu, 3-6
Worldwide technical support, B-2
Wrap option
Find command, Variables window, 10-4
Find UIR Objects dialog box, 4-11